Universidad Carlos III de Madrid
www.uc3m.es

Grado en tecnologías de telecomunicación

Curso 2015-2016

Trabajo Fin de Grado

# Puesta en marcha, prueba y validación de escenarios de un sistema de gestión de comunidades de Crowdsensing con procesado en el nodo sensor

---


Autor: Mario Prieto Macías

Tutor: Daniel Díaz Sánchez


Leganés, 12 de Julio de 2016

# Resumen

En los últimos años, el crecimiento exponencial del uso de los móviles, y la creación de ordenadores de muy reducido tamaño, ha favorecido la aparición de distintas plataformas de C*rowdsensing.* Estas se aprovechan de la gran cantidad de estos dispositivos que se mueven por las calles o del poco espacio que ocupan, para obtener información de distintos sensores que se pueden colocar en estos aparatos. Información que puede ser muy útil debido a la alta capacidad de procesamiento y a la constante movilidad, lo que otorga datos muy variados, elemento clave en una plataforma de recolección de información.

Este proyecto se enfocará en la creación de una plataforma de *Crowdsensing*, en la que cada usuario pueda crear su propio código con el que extraer la información necesaria en el momento en el que él desee. Y no solo esto, debido a las diferentes características de los distintos sensores, también será posible la elección de que sensores se desean utilizar, siendo posible seleccionarlos por: localización, tipo de sensor, etc. Esta propiedad, poco, o nada, usada en este tipo de comunidades, genera una gran atracción para usuarios que en las mencionadas plataformas sólo podían obtener datos de forma común para todos, haciendo que para algunos usuarios, fuera innecesaria tanta información y para otros se quedara corta.

El objetivo de una plataforma de este tipo es poder consultar los datos que otorgan los experimentos, por lo que se veía necesario, la creación de un front-end, al que se le ha dedicado la gran parte del tiempo de este proyecto de final de grado.

La ausencia de una estandarización en el mundo del C*rowdsensing*, supuso grandes problemas a la hora de la lectura de los datos de estos sensores y de las conexiones entre el usuario y la central que interconectaba todos los sensores, llamada *Clearing House*.

Debido al auge del uso de móviles y tabletas, el diseño se realizó de forma responsiva, es decir, adaptable para cualquier tamaño de pantalla. Este diseño, favorece el uso de la plataforma, pudiendo consultar los datos obtenidos desde cualquier lugar y en cualquier momento, sin la necesidad de usar un ordenador.

# Abstract

Crowdsensing's platforms have started to appear at last years because everybody uses a mobile and computers that occupy a little space. That platforms use mobiles which are moving in the streets to obtain information given by a sensor that mobiles and computers have. That information can be very useful because those mobiles have a big processing capacity and mobility. Data will be varied and this is an important point on a platform that collects information.

This project will be focus in the creating of a Crowdsensing's platform. Every user will can create his own code to extract data when he wants. But it's not the only skill that this platform has, It will be possible to chose which sensor will be in the experiment. Sensors will be selectable by his skills: location, type, etc. Other platforms don't have this ability because only can extract data in a prefixed time or location.

When someone uses a platform like this, wants to consult which is the information that experiments extract. To do this, it's necessary to create a front-end, the target of this project. The absence of a standardization made this project more difficult than the original idea. It was necessary to create a way to read data form sensors and send this data between user and the central, called: Clearing House.

Due to the rise of the use of the mobile devices, the design of the front-end was made responsive, it can fit to all screens. This design improves the use of this platform because users can consult data every moment, every time without a personal computer.

# Índice de figuras

Ilustración 1. Diagrama funcionamiento MVC ..... 25
Ilustración 2. Esquema niveles plataforma de *Crowdsensing* ..... 30
Ilustración 3. Características del diseño ..... 35
Ilustración 4. Esquema funcionamiento hash ..... 37
Ilustración 5. Esquema funcionamiento recepción peticiones ..... 42
Ilustración 6. Diagrama autenticación correcta ..... 49
Ilustración 7. Diagrama autenticación incorrecta ..... 49
Ilustración 8. Autenticación de la aplicación ..... 50
Ilustración 9. Diagrama funcionamiento Vaadin Designer ..... 54
Ilustración 10. Muestra información del usuario ..... 56
Ilustración 11 Muestra información de los experimentos ..... 57
Ilustración 12. Diagrama funcionamiento filtros en Java 8 ..... 58
Ilustración 13. Diagrama de Gantt ..... 67
Ilustración 14. Niveles plataforma Crowdsensing (inglés) ..... 79
Ilustración 15. Esquema funcionamiento aplicación (inglés) ..... 81
Ilustración 16. Diagrama funcionamiento Vaadin Designer (inglés) ..... 82
Ilustración 17. Diagrama funcionamiento HTTPS ..... 83

# Índice de tablas

Tabla 1. Tareas planificadas ..... 66
Tabla 2. Tareas planificadas (inglés) ..... 78

# Índice de contenido

1.Introducción ..... 9
1.1 Motivación ..... 9
1.2 Motivation ..... 10
1.3 Objetivos ..... 10
1.4 Objectives ..... 11
1.5 Contenido de la memoria ..... 12
1.6 Content of the document ..... 14
2.Estado del arte ..... 16
2.1 Crowdsensing ..... 16
2.2 Dificultades del tratamiento grandes datos ..... 21
2.3 Desarrollo aplicaciones web ..... 23
3. Diseño ..... 28
3.1 Introducción ..... 28
3.2 Arquitectura de la plataforma ..... 28
3.3 Arquitectura del front-end ..... 31
3.4 Seguridad de la aplicación ..... 36
4. Implementación ..... 39
4.1 Introducción ..... 39
4.2 Framework usado para implementar el diseño ..... 39
4.3 Descripción de los módulos ..... 40
4.4 Módulo de logs ..... 43
4.5 Módulo de información y valores ..... 46
4.6 Autenticación ..... 48
4.7 Base de datos ..... 50
4.8 Navegación de la aplicación web ..... 53

4.9 Comunicación con la base de datos ..... 58
5. Pruebas ..... 63
5.1 Pruebas de carga sobre el Clearing House ..... 63
5.2 Pruebas de funcionamiento de la aplicación ..... 64
6. Planificación y entorno socioeconómico ..... 66
6.1 Planificación ..... 66
6.2 Entorno socio-económico ..... 67
7. Conclusiones y líneas futuras de trabajo ..... 70
7.1 Conclusiones ..... 70
7.2 Conclusions ..... 71
7.3 Líneas futuras de trabajo ..... 73
Bibliografía ..... 74
Anexo ..... 76

# 1. Introducción

## *1.1 Motivación*

La posibilidad de estar comunicado en cualquier lugar del mundo, unido al auge en el uso de los dispositivos móviles, ha vuelto a sacar a la palestra la vieja idea de la creación de una gran red de sensores desechada hace años por los altos costes de computación y el gran tamaño que ocupaban los sensores. En la actualidad esta idea se ha vuelto realizable con las posibilidades actuales de computación en dispositivos tan pequeños como puede ser un móvil o una *Raspberry.*

La vieja idea de la red de sensores, ha sido recogida y convertida en las actuales plataformas de *Crowdsensing*, en la que los usuarios ceden el uso de sus sensores para crear una gran comunidad. Actualmente existen varias de estas plataformas, algunas incluso impulsadas por grandes marcas, pero cada una de ellas, usando su propia estructura de comunicación y de mensajes y recolectando los datos en períodos y en lugares predefinidos y fijos, sin posibilidad de elegir un intervalo de tiempo o de localización distinto al prefijado en el inicio de la plataforma.

El grupo Pervasive de la UC3M, desarrolló una plataforma en la que era posible la configuración de los periodos en los que se iba a recolectar datos de los sensores y que tipo o tipos de sensores se querían usar en los experimentos. El objetivo de dicha plataforma, era que los usuarIos pudieran acceder a dichos datos, por lo cual se veía necesario la creación de un front-end, que se comunicara con la central de la plataforma para recibir los valores y estado de los sensores; y enviara la información de los experimentos que los usuarios creasen.

En este documento, se relata tanto el diseño inicial de dicho front-end, tanto en la parte interna, como la parte visual, con la que el usuario interactuará, como la implementación final que se ha realizado, creando un prototipo con una funcionalidad básica.

## *1.2 Motivation*

Many years ago, some experts had an idea. They wanted to create a big sensor's network, but was rejected for the high computational cost and the great space that these sensors occupy. Now, that idea can be re-lived because technology has reduced those costs and that space with mobiles or Raspberries.

That idea has been converted to the actual Crowdsensing's platforms, where users yield his sensors to create a community. At present, there are a lot of this platforms and some has been stimulated by big companies. Every company uses his own communication's network with special messages. And it's impossible to change the period of extraction and the location of this.

Pervasive's group of the University Carlos III, developed a platform where it's possible to change some configuration of the extraction. It's possible to select which sensors will be in the experiments, too. The goal of this platform is that users can visualize data of this platform, it made necessary to develop a front-end. This front-end create a connection with the Clearing House to receive values and state of sensors.

This document contains the original design of the front-end (internal and visual) and the design's implementation that has been made, creating a prototype with a basic function.

## *1.3 Objetivos*

Este proyecto se integra dentro de una plataforma de Crowdsensing desarrollada por el grupo *Pervasive* del departamento de Telemática de la

Universidad Carlos III. El objeto de trabajo es la realización de un front-end completo en el que el usuario puede comunicarse con la plataforma y visualizar los resultados de los experimentos que haya creado.

Con todo ello, se establecieron unos objetivos principales:

- ✓ Estudio del mercado actual de plataformas de *Crowdsensing.*
- ✓ Investigar las comunicaciones, en búsqueda de una estandarización en las plataformas existentes.
- ✓ Diseñar una aplicación, muy intuitiva, en la que el usuario pueda consultar fácilmente tanto todos los datos de los distintos experimentos que ha realizado como realizar nuevos experimentos o consultar los logs de los mismos, de forma responsiva para que pueda ser usada en cualquier dispositivo.
- ✓ Prueba y depuración de la aplicación y del envío y recepción de los distintos datos.
- ✓ Redacción de una memoria en la que plasmar tanto el diseño original como la implementación final realizada junto con un capítulo con las líneas futuras que se pueden seguir para completar este proyecto.

## *1.4 Objectives*

This project is integrated inside a Crowdsensing's platform developed by the Pervasive's group of the Telematic department in the University Carlos III. The target of this project is the creation of a complete front-end in which one user can communicate with the platform to visualize the results of experiments that they have developed.

A few principal aims were established:

- ✓ Study of the currents Crowdsensing's platforms.
- ✓ To investigate communications to search a standardization in the existing platforms.
- ✓ To design a simple application in which one user can check all data of different experiments which has been created or create a new of this.
- ✓ To depurate app and all the communication.
- ✓ To redact a document where explain the design and the implementation. This documents will have a chapter with a futures lines to continue the project.

## *1.5 Contenido de la memoria*

En este capítulo se explica cuáles son las partes que forman esta memoria, describiendo de forma breve el contenido de cada uno de los capítulos:

- Introducción: contiene la introducción y los objetivos de este trabajo de fin de grado.
- Estado del arte: este capítulo está formado por 3 apartados, en los que se explican la situación actual de los mercados en los que se puede incluir el objeto de trabajo de este proyecto.

  Un primer apartado que habla de la situación del Crowdsensing en la actualidad, con ejemplos de plataformas.

Un segundo apartado que relata cual son los problemas que existen en el tratamiento de grandes cantidades de datos y como se puede resolver, con ejemplos de herramientas que se usan.

Un último apartado que explica cuál ha sido, y es, la línea de la programación de aplicaciones web, relatando cuales son las diferencias entre las distintas opciones.

- Diseño: se explica el diseño inicial que se realizó de forma teórica de un front-end como este. Relatando cuales deberían ser las características que deberían ser implementadas para conseguir un front-end completo. Junto a ello, se comenta las necesarias configuraciones de seguridad, tanto de la aplicación como de las conexiones con la plataforma de *Crowdsensing*, la cual también es explicada en este capítulo.

- Implementación: este capítulo se ha subdivido también en distintos apartados en los que se explican cuáles son los módulos finalmente desarrollados y las funcionalidades que tienen cada uno de ellos. La autenticación de la aplicación y las distintas tablas que formas la base de datos también forman parte de este capítulo que finaliza con la explicación de cómo se ha implementado la navegación dentro de la aplicación y las distintas opciones que el usuario tendrá para elegir.

- Pruebas: En este capítulo se explican cuáles han sido las pruebas realizadas para la comprobación del correcto funcionamiento del proyecto.

- Planificación y entorno socioeconómico: este capítulo muestra cual ha sido la planificación de las tareas del proyecto, mostrando sus tiempos y cuál es el entorno socioeconómico en el que se engloba este trabajo de fin de grado.

- Conclusiones y líneas futuras de trabajo: es el capítulo que finaliza este documento, en el que se explican cuáles son las líneas que podrían continuarse en el futuro para completar este proyecto. También están incluidas en este capítulo las conclusiones que el alumno ha extraído tras la realización de este proyecto.

## *1.6 Content of the document*

This chapter explains which parts constitute this document, describing the content of every chapter:

- Introduction: contains the introduction and targets of this bachelor's thesis.

- State of art: this chapter is formed by 3 paragraphs which explain the current situation of business.

  A first paragraph which explain the Crowdsensing situation,with examples.

  Problems of the computing about a big amount of data are explained in the second paragraph. Tools to solve this problems are described in this paragraph, too.

  The last paragraph includes which are the different ways to program a web application.

- Design: explains the initial design of a front-end. This chapter describes properties to implement a complete front-end and the necessary security configurations in the application and in the platform.

- Implementation: this chapter has a few paragraphs, too. These paragraphs explain which modules were developed and what functions have every one of this. Authentication and tables of

database are included in this chapter that ended with the implementation of the navigation and the different options.

- Test: explains what has been the analysis of the application testing the common use of this.
- Planning and socio-economic environment: this chapter shows which has been the planning of the project and the socio-economic environment of this project.
- Conclusions and future lines of work: the last chapter explains which are the lines everyone can continue to complete this project. Conclusions of the student are included in this chapter, too.

# 2. Estado del arte

## *2.1 Crowdsensing*

Hace 15 años, nacía la plataforma *Wikipedia,* definida por sus creadores como: "*una enciclopedia digital de acceso libre"*. Pero, no es solo el acceso lo que es libre, también lo es la edición de las definiciones o artículos que componen esta enciclopedia, que fue el séptimo sitio web más visitado en el año 2015, solo por debajo de súper potencias como Google, Youtube o Facebook. La idea original fue la creación de una enciclopedia escrita, llamada *Nupedia* **[1]***. A*cadémicos y grandes especialistas debían dar el visto bueno para que los artículos fueran publicados, este proceso era demasiado lento y por ello se creó, de forma paralela, *Wikipedia,* en la que cualquier persona, puede escribir, corregir, o editar artículos.

Este éxito, fomentó la aparición de servicios que seguían la misma filosofía, los usuarios de forma altruista, ayudaban a crear una pila de datos que pudieran reportar información útil para un fin. En el MIT ***(Massachusetts Institute of Technology)***, desarrollaron esta idea para crear una aplicación, que basada en la información que los usuarios mandaban sobre su ubicación y velocidad, creaba rutas de tráfico óptimas, evitando en gran medida, la aparición, o al menos el aumento, de los atascos típicos en grandes ciudades. Para evitar el consumo de recursos en los dispositivos de los usuarios, y que este proceso fuera lo más transparente posible, CarTel, nombre de esta plataforma, usa los puntos de acceso Wi-Fi a los que se conecta durante su recorrido, aunque sea solo por segundos, para conectarse con la central que tramita todos esos datos, un sistema novedoso que junto con un algoritmo de creación de rutas, compone una aplicación útil, y realmente novedosa.

Otro gigante como IBM, y dentro de la misma línea del MIT, desarrolló una aplicación para iPhone, usada para monitorizar la calidad de las aguas del

estado norte americano de California. Su funcionamiento es muy sencillo, los habitantes de la zona, se pueden acercar a los ríos y realizar una foto para enviarla mediante la aplicación junto con algo de información acerca de la cantidad de basura encontrada en el agua, el rango de fluidez de la misma y la cantidad que había. Esta información es recibida por las autoridades encargadas de cuidar el agua del estado y crean una base de datos con la que poder saber cuál es la situación de los distintos ríos.

La llegada de las nuevas tecnologías ha cambiado el mundo en el que vivimos, tres cuartas partes de las personas que habitan este mundo usarán diariamente un teléfono móvil en 2020, la conectividad entre humanos es total. Pero la revolución no se ha querido quedar ahí, esta conectividad ha llegado hasta el mundo de las máquinas, ya no solo estamos conectados con nuestros amigos, compañeros de trabajo o familiares, ahora también lo estamos con nuestra lavadora, nuestro frigorífico o el alumbrado de nuestra casa.

La llegada de IPv6, impulsó, más aún, este auge. Con la anterior versión de IP (versión 4), las distintas direcciones tenían un límite, al cual estábamos a punto de llegar. Esta nueva versión, con un direccionamiento distinto, provoca que la cantidad de direcciones IP otorgables a los distintos dispositivos sea enorme, en palabras de Steve Leibson: “Si se diese una dirección IP a cada átomo de la tierra, con IPv6, podríamos tener más de 100 tierras con una dirección IP por átomo” **[2]**. En resumen, altamente imposible el volver a colapsar el direccionamiento IP. Por ello, a cada dispositivo interconectado, es posible otorgarle una dirección IP, y realizar toda la conectividad bajo un protocolo completamente expandido y estandarizado, facilitando así, de gran manera, la dificultosa y aun inacabada estandarización de la conexión M2M (máquina a máquina). Con la extensión del uso de los móviles y de la conectividad entre máquinas, se ha resucitado viejas ideas acerca de la creación de redes de sensores para obtener diversos datos, ya sean del ambiente natural, o del artificial, como las aplicaciones mencionadas con anterioridad. Estas ideas, se

están empezando a desarrollar y hay expertos en la materia que intentan, como se ha dicho sin mucho éxito, crear estándares, o al menos recomendaciones.

Muchos de los creadores de estas plataformas de redes de sensores que recolectan datos, hablan de 5 niveles que este tipo de plataformas debieran tener **[3]**:

1. Nivel Físico

   Este nivel es donde se sitúan los sensores, pueden ser sensores embebidos dentro de los dispositivos, o pueden ser externos, conectados a los mismos, como sensores de humedad o temperatura en *RaspBerrys*. Creando una red de sensores conectados entre sí mediante los dispositivos.

2. Recolección de datos

   Los datos obtenidos son transferidos desde los sensores, nivel físico, a los dispositivos en sí. En muchas ocasiones, como en la toma de medidas de humedad o temperatura, la información obtenida es analógica, por lo que se debe convertir a digital para poder trabajar más tarde con ella.

3. Trabajar los datos

   Los datos obtenidos, han de ser transformados en datos inteligentes, es decir, datos de los cuales se puede obtener información que pueda ser mostrada al usuario. Es típico el uso de algoritmos que generen estos datos de forma fácil y sencilla. Este nivel, puede ser realizado en los dispositivos, por ejemplo con el uso de actores, o puede ser realizado en unas centrales, que reciban todos los datos en bruto y sean las encargadas de pulirlos y obtener algo útil, por ello,

según como el creador de la plataforma desee, este nivel puede ser intercambiado con el siguiente.

4. Envío de datos

Estos datos, una vez han sido procesados en los dispositivos, son enviados a una central que interconecta todos los dispositivos de la red. Hay muchas redes para el envío de estos datos: Wi-Fi, Bluetooth, la red movil, y muchos protocolos para la misma, pudiéndose usar HTTP, FTP, etc… Este envío, debe gastar los menos recursos posibles en los dispositivos para que el usuario no vea afectado este envío en su uso diario.

5. Muestra de datos

La verdadera finalidad de una plataforma de este estilo es que empresas, organismos o particulares, puedan consultar los distintos datos que se han obtenido. Por ello, una vez se han procesado, se deben mostrar al usuario, mediante una aplicación, o una interfaz web. La comunicación con la central, que tiene todos los datos puede, de nuevo, realizarse al gusto del desarrollador.

Como vemos, una plataforma de *Crowdsensing,* tiene unos niveles comunes, pero es casi lo único que tienen en común. Como se ha comentado con anterioridad, hay una gran ausencia de estandarización, el desarrollador puede elegir entre miles de protocolos y formas para comunicarse, lo que dificulta mucho el trabajo. Es cierto que hay organizaciones que intentan crear un

estándar, pero debido, con mucha probabilidad, a intereses comerciales, cada marca, o organización, genera su propia comunicación. Esta ausencia de uniformidad en las comunicaciones, provoca que sea muy complicado la creación de nuevas redes de *Crowdsensing,* puesto que no solo hay que desarrollar la idea inicial, también como se van a realizar los envíos de datos entre los sensores y la central, y de esta con el usuario final.

La idea original de este tipo de plataformas, era la ayuda desinteresada de los usuarios que cedían sus dispositivos sin esperar nada a cambio. Pero en general, cualquier ser humano, se siente más favorable a hacer algo cuando hay un incentivo de por medio. Estos incentivos, no siempre tienen por qué ser económicos. Es posible la retribución mediante la concesión del uso de la plataforma de forma limitada y gratuita, con ello se puede incentivar a que las personas cedan sus dispositivos de forma más extendida. Además estos usuarios necesitan saber que su información va a ser usada con seguridad.

Estas plataformas, obtienen datos muy sensibles de los usuarios, saben dónde están en todo momento con el GPS, o podrían averiguar donde vive una persona con saber que ubicación se repite en su listado. Por ello uno de los puntos primordiales de este tipo de redes de sensores es la seguridad de los datos y el compromiso a la confidencialidad **[4]**. Los datos, en todas las comunicaciones, deben ser cifrados para que nadie que haya interceptado esas comunicaciones pueda obtener los datos en claro. Como cualquier comunicación en la actualidad, esta misma también debería ser segura, usando protocolos que establezcan medidas de seguridad como pudiera ser HTTPS. Pero no solo se debe cifrar los datos durante las comunicaciones, el almacenaje de los mismos ha de ser de igual forma seguro, para que si un atacante, intenta penetrar en la base de datos, le sea muy difícil poder saber cuál son los datos de la misma.

## *2.2 Dificultades del tratamiento grandes datos*

Los humanos creamos gran cantidad de información a mano, según diversos estudios, todas las obras escritas a mano por la humanidad en los diferentes idiomas existentes, podrían recogerse en 50 petabytes de datos, esta cantidad, se queda en nada comparado con la que Google, una sola empresa, trabaja cada día, unos 20 *petabytes*, es decir, en un día se tramitan a máquina algo menos de la mitad de todas las obras de la historia escritas a mano. Por estas limitaciones, la idea de la creación de una red de sensores lleva mucho tiempo en la cabeza de los investigadores que por el alto coste computacional y monetario, veían como esta idea debía ser aparcada. Pero con la situación actual de las tecnologías ha cambiado las tornas, la computación distribuida puede resolver el problema de la gran cantidad de datos que se debían procesar en una red como la pensada, y no solo por la cantidad, sino por la gran rapidez y facilidad.

En el "**Manifiesto de Sistemas Reactivos"** **[5]**, desarrolladores, líderes de proyectos y en general expertos de la materia, hablan de este cambio, comparando el número de servidores y computadoras y los tiempos de respuesta que las aplicaciones tenían hace no muchos años con la situación actual, en la que hasta el mas mínimo dispositivo es capaz de desplegar una aplicación muy potente y con mínimos tiempos de espera para el usuario. En este mismo manifiesto se habla acerca de como deben ser los sistemas de computación actuales, en concreto hablan de 4 características básicas:

1. Responsivo. Rápidos tiempo de respuestas
2. Resilente. Alta disponibilidad.
3. Elásticos. Resisten a grandes cambios de carga
4. Orientados a mensajes. Comunicación no bloqueante e intercambio asíncrono.

Dentro de las diversas posibilidades, el modelo de actores, encaja a la perfección en las propiedad de las que el manifiesto habla, un modelo con un funcionamiento muy sencillo: se recibe una comunicación del exterior, una tarea que se ha de realizar. Esta tarea es recibida por el actor maestro, quien la subdivide en pequeñas tareas que son realizadas por los diversos actores que componen el sistema, en el cual cada uno de estos actores, pueden realizar tres funciones básicas por cada nueva tarea entrante:

1. Enviar mensajes a otros actores

2. Cambiar de comportamiento, es decir, realizar una tarea distinta a la que estaba realizando.

3. Crear un nuevo actor para que realice una tarea.

El actor maestro controla cuál de los actores debe ser el siguiente en participar, y así se crea un mecanismo sincronizado en el cual cada actor realiza su función única y exclusivamente cuando el maestro así se lo ordena. Dentro de la misión encargada por el maestro, el actor puede realizar las funciones mencionadas más arriba. Cuando la tarea ha sido finalizada, los actores creados pueden ser eliminados, o pueden mantenerse para futuras tareas que se vuelvan a requerir.

Como vemos, el modelo de actores, cumple con los 4 requisitos del manifiesto, y es por ello que es uno de los modelos más usados, siendo *Erlang* uno de los primeros lenguajes con el que se podía implementar este modelo. Apareció en 1986 a manos de la compañía Ericsson pero en el año 1998 fue sacado al público general como software de código libre. El funcionamiento de Erlang fue y sigue siendo magnífico, y en la actualidad, siguen siendo lanzadas al mercado versiones actualizadas, pero la dificultad bastante alta para programar en este lenguaje, favoreció la aparición de distintas herramientas que reducían esta dificultad basándose en Erlang, siendo *Akka*, una de las más extendidas.

Esta herramienta, nacida en 2009, con una capacidad de computo enorme soporta varios lenguajes de programación, lo que amplía las posibilidades para los desarrolladores, siempre basándose en el modelo de actores comentado con anterioridad, con la posibilidad de enviar mensajes no bloqueantes y asíncronos. Además, los procesos son muy ligeros, aproximadamente y según la misma compañía 2,7 millones de actores por 1 GB de RAM **[6]**. *Akka*,cumple con los principios que en se explican en el manifestó y es una herramienta cuyos procesos son transparentes para el usuario y tolerante a fallos, con posibilidad de replicar en varias máquinas virtuales para poder trabajar en caso de caída o sobrecarga de una de ellas. Es posible programar en Java y en Scala usando Akka, siendo muy útil la posibilidad de Java debido al extenso uso de este lenguaje en la gran mayoría de desarrolladores.

Con una herramienta como esta, el problema que aparecía en el momento de querer crear una red de sensores, debido a la dificultad de trabajar con grandes datos, es fácilmente solucionable, pudiendo incluso trabajar con dichos datos en los mismos dispositivos, dada la ligereza de los procesos creados para procesar dicha información.

## *2.3 Desarrollo aplicaciones web*

Las plataformas de *Crowdsensing*, necesitan que los datos que los sensores obtienen, sean tramitados y trabajados, y esa es la función de la computación distribuida, pero una vez todos esos datos se han procesado, deben ser mostrados al usuario final, que es a quien realmente está enfocado este proceso. Estas plataformas suelen tener una central, que es la que conecta todos los dispositivos entre sí y con el usuario final.

En la actualidad existen diversos métodos para crear aplicaciones web, la gran mayoría basados en el modelo cliente-servidor en el que el servidor, que puede ser más de uno, pone a disposición del cliente, diversos archivos, o elementos,

que el cliente debe solicitar al servidor para que este se los prepare y puedan ser mostrados al usuario. Toda la tecnología alrededor del desarrollo web, ha ido evolucionando a lo largo de los años. Una de las primeras tecnologías, que se usaron fue los Servlets, de Java, usados para responder a peticiones que se realizaban a un servidor, ejecutando el método *init*, cuando una petición es recibida en el servidor. Usa el modelo de programación petición-respuesta, es decir, se reciben peticiones en el servidor, que tras ejecutar el flujo de funcionamiento, devuelve una respuesta a quien origino la petición. En su formación original, solo servía contenido estático, por lo que se fue desarrollando mejoras, que permitieran distribuir en varios servidores web la aplicación además de mostrar contenido dinámico, siendo JSP (Java Server Pages), creada por Oracle también, la principal impulsora de este cambio.

El motor de las páginas está basado en los *servlets* originales, sirviendo archivos con extensión ".jsp", que incluyen dentro contenido en html y sentencias en java que se ejecutan en el servidor, estos archivos son procesados, en ocasiones de forma previa, en otras al recibir la primera petición, y convertidos en servlets, lo que hace que las peticiones sean finalmente procesadas por uno de ellos. Estos archivos, pueden estar expresados en formato estándar, o en XML, pero solo en uno de los dos, nunca mezclando. Esta tecnología, facilitaba la tarea de diseñar la aplicación web, pero de nuevo, la constante evolución, ha hecho quedarse un tanto atrasada a esta tecnología, siendo desarrollados nuevos patrones, con mejores características que se fueron extendiendo **[7]**.

Uno de estos patrones de desarrollo web, de los mas usados, es el modelo vista controlador (**MVC**), el cual separa la lógica de negocio en tres niveles:

1. Vista. Pueden ser varias, es la parta gráfica, la interfaz con la que el usuario interactúa, suele estar escrita en HTML, un lenguaje de programación web.

2. Controlador. De igual manera, pueden co-existir varios, es el encargado de unir, la parte visual con el modelo, obtiene datos de forma dinámica y los genera, de la misma forma.

3. Modelo. Es una representación de los datos de la aplicación, el controlador, obtiene del modelo los datos que luego serán mostrados al usuario, estos datos pueden ser modificados y ser de nuevos guardados en el modelo.

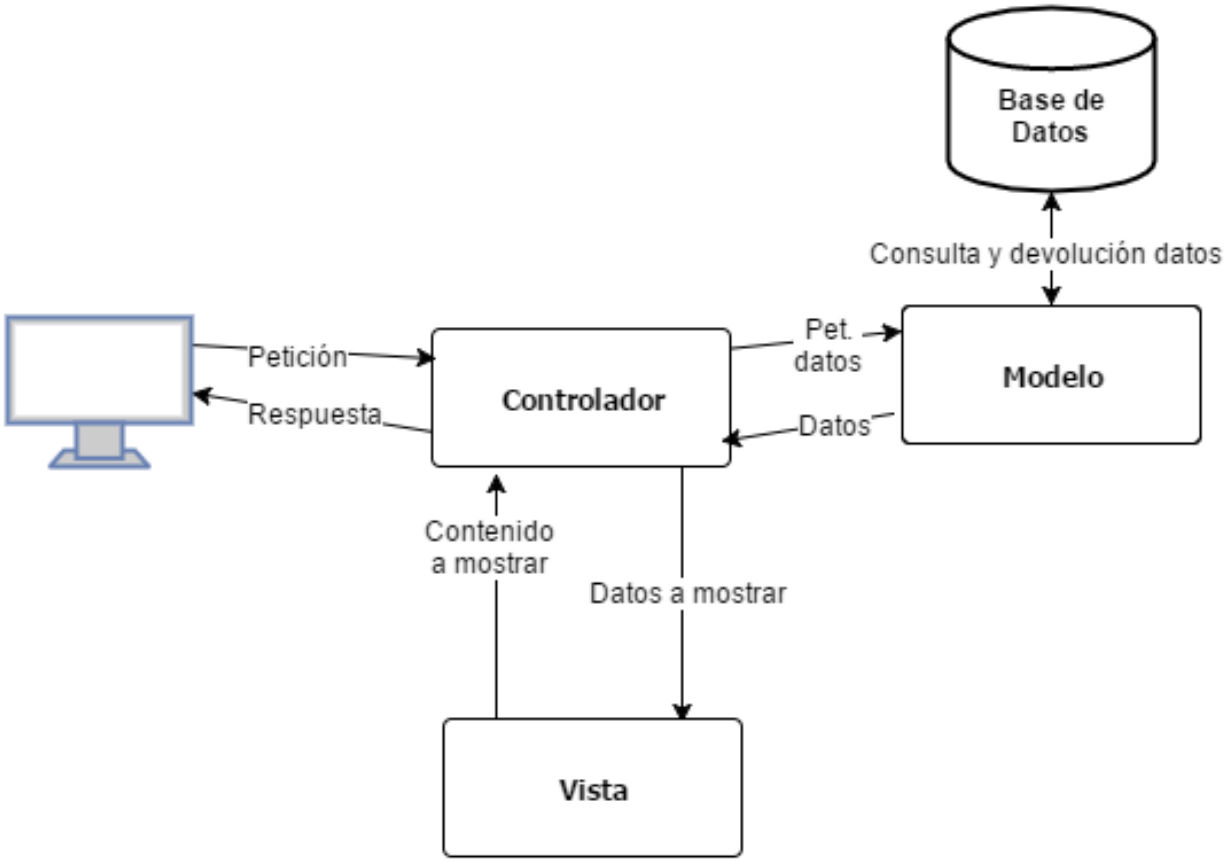


**Ilustración 1. Diagrama funcionamiento MVC**

Originalmente, el patron MVC, estaba diseñado para aplicaciones de escritorio, se ha expandido su uso en el desarrollo de aplicaciones web provocando que

fueran lanzados al mercado multitud de frameworks basados en este patrón como *Ajax*, *Spring* o *AngularJS.*

Con el tiempo, este patrón fue evolucionando, y nuevos estilos están copando en mercado por su facilidad para ser implementado por los desarrolladores en futuras aplicaciones, REST es uno de estos ejemplos **[8]**. Este estilo no está relacionado siempre con HTTP, pero si lo está en la gran mayoría de ocasiones. Tiene como diferencia con MVC, la forma en la que las respuestas HTTP son creadas. Es un estilo sin estado, cada petición que se realiza no mira mas allá de si misma, es decir se pide algo sin importar además, si hay alguna restricción en la red o en el servidor. La devolución a estas peticiones suelen ser documentos, XML, JSON o incluso HTML, estos documentos deben ser comprendidos por la aplicación para poder procesarla y trabajar con ella. REST, otorga a los desarrolladores distintos métodos para manejar las peticiones HTTP, desde solo consultar hasta editar estas peticiones, pasando por eliminar o crear. La curva de aprendizaje de este patrón es muy reducida en comparación con otros patrones de desarrollo web, lo que provoca, que sea muy sencillo crear aplicaciones usando REST. Actualmente, la gran mayoría de aplicaciones, como Twitter, DropBox, o Facebook tienen implementado una API basada en REST con el que poder interactuar para crear una aplicación alternativa a las oficiales de forma fácil y sencilla.

Dentro del desarrollo web, se han lanzado al mercado diversos frameworks para hacer más fácil el trabajo a los desarrolladores, uno de ellos, Vaadin **[9]**, fue lanzado al mercado en 2009, y desde entonces no ha dejado de crecer, tanto en usuarios como en funcionalidad complementarias a la original. Este framework, basado completamente en JAVA, unifica todo el ámbito del desarrollo web en un entorno fácil, debido al solo uso de un lenguaje de programación. Vaadin, saca al desarrollador del bajo nivel, haciendo ese trabajo por el, y solicitando solo el uso de objetos (principio básico de Java), para la creación de elementos. Los cambios en la interfaz que el usuario realice, afectan a estos objetos, los cuales tienen adheridos *listeners,* que

capturan esos cambios y con una lógica creada, cambian la interfaz de cara al usuario. Estas particularidades, son un gran avance para los creadores de aplicaciones web, pues de primeras, se elimina la diversidad de lenguajes necesarios para crearla (HTML, JAVA, CSS, etc.), y no solo eso, también tiene una gran sencillez en el momento de crear lógicas de comportamiento. La parte del diseño, quizás la más tediosa a la hora de crear una aplicación, es resuelta con facilidad con uno de los Add-On que la empresa ha lanzado, llamado *Designer*, y con el cual, con una lista de elementos, que pueden ser arrastrados en un lienzo que simula la pantalla de un ordenador, tableta o dispositivo móvil. Por todas estas características Vaadin es uno de los frameworks más usados y con más proyección del mercado.

# 3. Diseño

## *3.1 Introducción*

En este capítulo se expone cual es el diseño que un front-end integrado en una plataforma como esta debería tener, explicando cual es el funcionamiento de la plataforma y cuáles son las comunicaciones que esta tendrá con el front-end, incidiendo en los distintos módulos de los que se dispondría para poder acceder a todas las características de la plataforma.

Esta aplicación, debe ser de muy sencillo uso para el usuario, intuitiva y, a poder ser, mutidispositivo, para poder ser funcional en cualquier aparato que hoy en día usen los usuarios. La plataforma tramitará toda la comunicación mediante el *Clearing House*, una central que es el núcleo de la plataforma y que recoge toda la información.

## *3.2 Arquitectura de la plataforma*

Como se ha explicado, la plataforma en la cual se va a integrar este front-end, es una plataforma de *Crowdsensing*, que se basa en la recolección de información por parte de dispositivos con sensores conectados. Estos dispositivos, pertenecen a usuarios y pueden ser desde un teléfono móvil, hasta una mini computadora como una *Raspberry.*

Estos dispositivos, son cedidos a la plataforma por parte de los usuarios, y son registrados por el *Clearing House*, que es una central, que está conectada a todos los dispositivos que integran la plataforma, recibiendo y enviando mensajes a los mismos. Estos mensajes son muy diversos: los dispositivos pueden enviar mensajes con los valores de sus sensores, o con información acerca de los mismos, como puede ser que uno de ellos se haya estropeado, recalentado, o desconectado; por otra parte el *Clearing House* puede

comunicarse con los dispositivos para informar de nuevas situaciones de la plataforma, y para enviar información de los nuevos experimentos, que es en lo que se basa esta plataforma.

Estos experimentos, son programas creados por los usuarios del sistema, en el cual configuran que tipo de sensores desean, cada cuanto tiempo quieren que estos sensores recojan información, además de seleccionar, si se desea, que la información sea recogida solo en algunas zonas geográficas, como pudiera ser un sensor que estuviera en un autobús, y que solo enviara información al *Clearing House* cuando entrara en una determinada zona de la ciudad, o cuando estuviera a una determinada altitud con respecto al nivel del mar, por ejemplo. Estos experimentos, serían programados por los usuarios, y mediante la web, subidos al sistema, para que el front-end los hiciera llegar al *Clearing House*, quien tramitaría dicho programa, y dependiendo de las características del mismo, enviaría la información a los dispositivos.

La novedad de este sistema, respecto a otros que existen, o han existido, es la autonomía de los usuarios respecto a los creadores de la plataforma, puesto que podrían crear experimentos que se adaptaran a la perfección a sus deseos, como los de situación geográfica o de tiempo, obteniendo los datos cada día o cada segundo, según el usuario los necesite.

El usuario, deberá poder obtener información acerca del funcionamiento de sus experimentos, por si alguno de los dispositivos que pertenecen a dicho experimento se ha estropeado o deja de funcionar correctamente. Esta información, se mostrará al usuario mediante la aplicación, en forma de logs, que serán enviados por el *Clearing House,* que al ser la central, tendrá toda la información de todos los dispositivos que pertenecen al sistema y puede crear estos logs para los distintos experimentos.

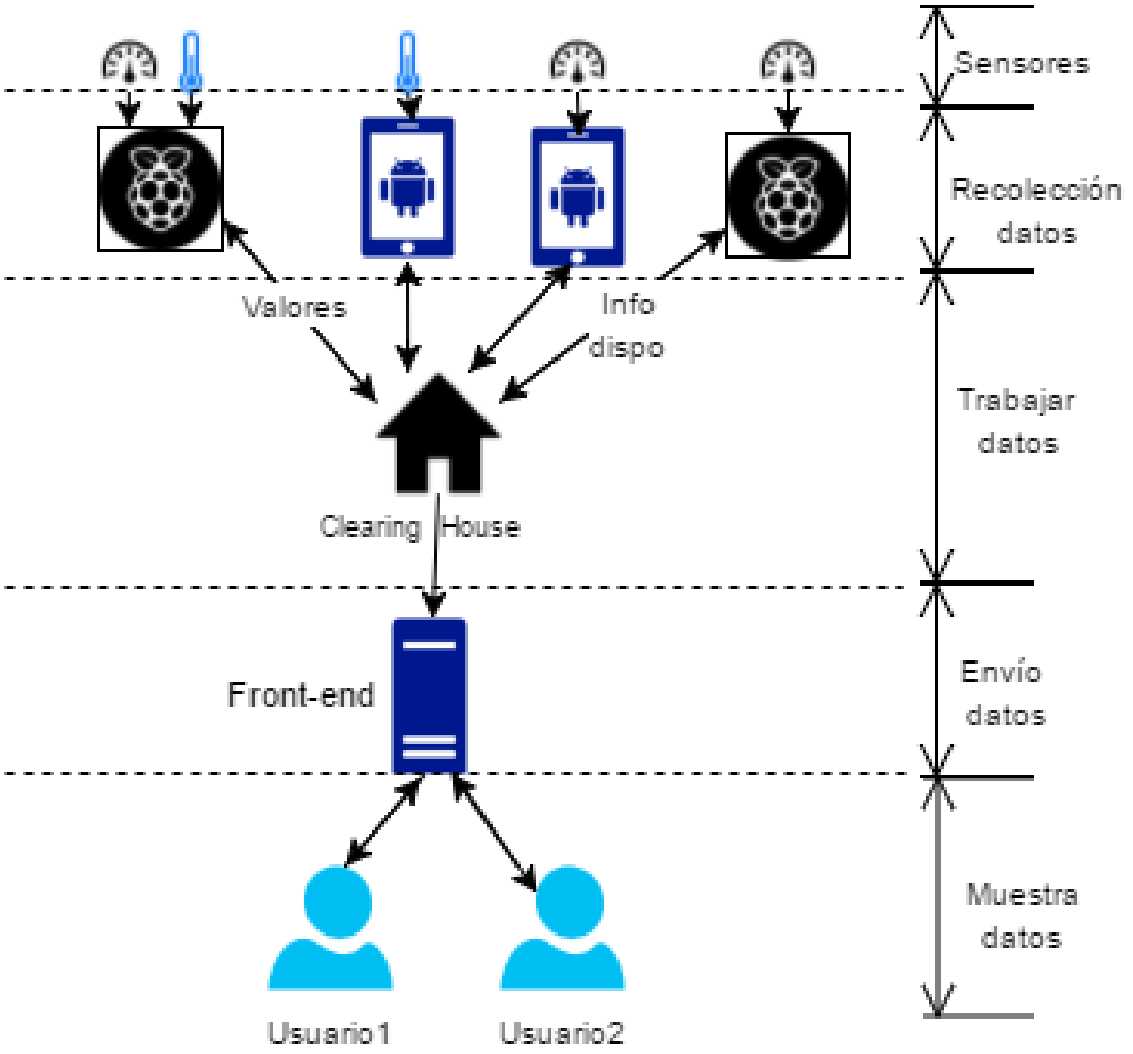


**Ilustración 2. Esquema niveles plataforma de *Crowdsensing***

Como vemos, la plataforma consta de los 5 niveles de los que se hablaba en el capítulo “Estado del Arte”, niveles que los expertos en *Crowdsensing* hablan que debe tener un sistema como este:

- Primer nivel, el nivel físico, formado por los sensores, que están conectados a los móviles o las *raspberrys*, quienes recogerán los datos que se mostrarán.

- Segundo nivel, formado por los dispositivos en si, que son los encargados de recolectar la información de los sensores y preparla para poder ser tratada luego.
- Tercer nivel, en el que se trabaja con los datos, en este caso los envuelve dentro de mensajes dependiendo de a que experimento pertenezca para poder ser luego enviados al siguiente nivel.
- Cuarto nivel, es un nivel en el cual se transmiten los datos en formato json, con identificadores, para poder ser luego leídos por el front-end, que es quien finalmente los recibe en este nivel, y quien los mostrará al usuario final de la aplicación.
- Quinto nivel, es el único nivel con el que puede interactuar el usuario, mostrándose la información que este desea ver, y también pudiendo enviar información de vuelta, como la descripción y el desarrollo del experimento que vaya a crear.

Estos niveles, no son ni mucho menos obligatorios en toda comunidad de *Crowdsensing*, pero si es una gran forma de crear una uniformidad en todas ellas, facilitando a los desarrolladores y usuarios el ingreso en dichas plataformas.

## *3.3 Arquitectura del front-end*

Dentro de esta plataforma, el usuario debe poder consultar diversa información acerca de la misma, esta información debe ser guardada en la base de datos de la que el front-end dispondrá para poder ser mostrada al usuario.

El *Clearing House* enviará de forma autónoma al front-end (sin que este pida nada) mensajes con dicha información. Estos mensajes tendrán distintos formatos, debido a que cada uno de ellos contiene datos distintos, por ello

tendrán que ser tratados también de forma distinta dependiendo de cual sea el mensaje que se reciba.

El front-end, dispondrá de distintos servlets, que encaminaran las peticiones según la url a la que sean mandadas teniendo cada uno de estos servlets una lógica que tratará los mensajes que irán dentro de estas peticiones. Por ello, este front-end, podrá recibir dos tipos de peticiones a grandes rasgos: un tipo que provendrá del usuario, debido a la interacción con la aplicación, y otro tipo que será el que se reciba cuando el **Clearing House** envíe las peticiones *POST*. Todas las peticiones del primer tipo, serán tratadas para que el usuario pueda comprobar la información, esta información se abstraerá de la base de datos en donde se habrá guardado previamente, los valores de los sensores, su situación actual (mediante los logs) e incluso la existencia de los mismos, para poder navegar correctamente por la aplicación web, el diseño de la misma debe ser muy intuitivo, sencillo y de fácil uso, además de adaptarse a cualquier tamaño de pantalla, para así conseguir llegar a una mayor cantidad de usuarios. Por ello, es recomendable el uso de algún framework de los disponibles en el mercado, como puede ser *Vaadin*, el cual ayuda a evitar al desarrollador descender a un bajo nivel en la programación.

Para poder tener todas estas funcionalidades relatadas, el front-end tendría distintos módulos:

1. **Módulo para los logs**. Este módulo debe recibir las peticiones que el *Clearing* House enviará, dentro de las cuales, se podrá obtener un JSON, con el identificador del experimento, y el log del mismo. Este módulo, leerá dicho JSON, extrayendo la información e insertándola en la base de datos para poder ser mostrada más tarde al usuario.

2. **Módulo para los experimentos**. Este módulo se encargará de enviar al *Clearing House* el experimento que el usuario ha creado, indicando que dispositivos componen dicho experimento y cada cuanto y en qué zonas, se debe obtener datos. Por ello, debe leer de la aplicación que ha

seleccionado el usuario, y convertirlo en un formato legible por el *Clearing House*.

3. **Módulo para información y registro de dispositivos**. Tanto cuando sea añadido un dispositivo nuevo en la plataforma como cuando uno de los ya existentes cambie de estado (dañado, apagado, poca batería, operativo, etc), se debe actualizar la base de datos con esta información para que el usuario pueda conocerla. Por ello, este módulo debe ser el receptor de los POST que el *Clearing House* enviará y de dichas peticiones poder obtener la actualización de la plataforma e ingresarla en la base de datos, en las respectivas tablas.

4. **Módulo para la cuota.** Este módulo será el encargado de gestionar y controlar el uso de la cuota otorgada a cada uno de los usuarios, limitando las acciones en caso de sobrepasarla e informar al usuario de cual es su nivel de ocupación en el momento. Esta cuota podrá ser aumentada o disminuida bajo diferentes factores que el usuario podrá modificar.

5. **Autenticación.** Los usuarios, deben identificarse al entrar en la aplicación para poder consultar los ratos relativos a su persona, y no a la de otra. Además cada usuario podrá tener una cuota ocupada, por ello se debe guardar en la base de datos toda la información tanto personal (nombres, correo, empresa, etc) como información relativa al sistema (cuota ocupada, experimentos creados, etc). Esta autenticación debe poder hacerse de distintas formas, una básica, en la que el usuario ingrese un nombre de usuario y una contraseña, que serán comprobados contra la base de datos; además de una autenticación mediante diversos servicios muy usados en la actualidad, como puede ser Gmail, Facebook o Twitter. Esta última forma, se puede implementar mediante el protocolo OAuth **[10]**, mediante el cual se puede autorizar a los usuarios de una forma muy sencilla y fácil.

6. **Navegación del usuario.** El usuario que use la aplicación podrá navegar por las diversas opciones que la aplicación tendrá, que mostraran información relativa a cada uno de los módulos de los que el front-end dispone. Esta navegación, interactuará con la base de datos, para poder mostrar la información. La navegación debe ser sencilla y muy fácil de comprender, sin tener que hacer muchos clics para acceder a la información que el usuario quiera consultar. En los apartados donde sea necesarios filtros, como puede ser la pantalla donde se muestran los dispositivos, dichos filtros deben ser lo más fáciles y rápidos posibles, mostrando los datos nada más el usuario escriba en dichos filtros. Esta navegación debe ser igual para cualquier dispositivo que se use, teniendo todas las funcionalidades, sea un móvil, una tableta o un ordenador.

7. **Base de datos.** El front-end deberá disponer de una base de datos en la cual almacenar toda la información que procesará proveniente del *Clearing House* para poder mostrársela al usuario cuando este quiera consultar dicha información. Esta base de datos, debe ser lo menos pesada posible, y con una alta velocidad de acceso a los datos para que le sea transparente al usuario. Las tablas que se vayan a crear en dicha base de datos, pueden contener valores de cadenas de texto como los logs, debido al tamaño reducido de estas cadenas. Se han de crear tablas que guarden:

   - ✓ Información personal del usuario (nombre, correo, nombre empresa, contraseña, etc)
   - ✓ Experimentos existentes en el sistema, indicando quien es el usuario creador y la situación actual.
   - ✓ Dispositivos existentes en el sistema, guardando que tipo de sensores tienen cada uno de estos dispositivos.

- ✓ Los logs que se hayan extraído de cada uno de los experimentos, indicando cual es el experimento al que hace referencia y cuál es el usuario creador de este experimento.

- ✓ Los valores de los sensores que el *Clearing House* habrá enviado y que se recolectan cada cierto tiempo, dependiendo de los la frecuencia programada por el usuario.

Estas tablas, deben contener al menos la información explicada en estas líneas para poder cumplir con las especificaciones, aunque es posible la creación de un mayor o menor número de tablas, con distintas columnas que pueden estar relacionadas unas con otras para obtener, finalmente la misma información.


**Ilustración 3. Características del diseño**

## *3.4 Seguridad de la aplicación*

Todos los datos que se guarden en la base de datos, deben ser tratados con la máxima seguridad posible, debido a que en ocasiones estos datos son de carácter muy sensible: alguien que pudiera acceder a los datos de posición de una persona, podría saber donde vive, trabaja o por donde se mueve.

Estos datos, deben ser cifrados cuando se insertan en la base de datos, mediante los diferentes métodos criptográficos actuales, o distorsionando las medidas tomadas, una técnica bien usada para no conocer la ubicación exacta de las medidas. Esta distorsión de la ubicación, como puede ser desviar unos metros la posición, al aplicarse al conjunto de todos los datos de la plataforma, no distorsionan los resultados que el usuario más tarde observará. Todo ello, con el fin de que si alguien consiga los datos, no pueda obtener la ubicación precisa de donde fueron tomados.

La seguridad de la aplicación no se puede quedar solo en el cifrado de los valores tomados por los sensores, los datos de los usuarios también son guardados en la base de datos, entre ellos la contraseña, valor que no debe ser guardado en claro dentro de las tablas, por ello, las contraseñas pueden pasar un proceso como puede ser una función de hash (MD5, SHA, *bycript*, etc). Estas funciones, generan una cadena de texto, llamada hash, que es la que se guarda en la base de datos, descartando la contraseña en texto plano. Cuando el usuario inserte la contraseña, por ejemplo en la autenticación para acceder a la aplicación, esta función de hash es aplicada a la contraseña que el usuario inserte, y se compara con el valor que se tenga guardado en la base de datos, en caso de ser iguales, la contraseña introducida es correcta, en caso de no ser igual, la contraseña introducida no es válida.

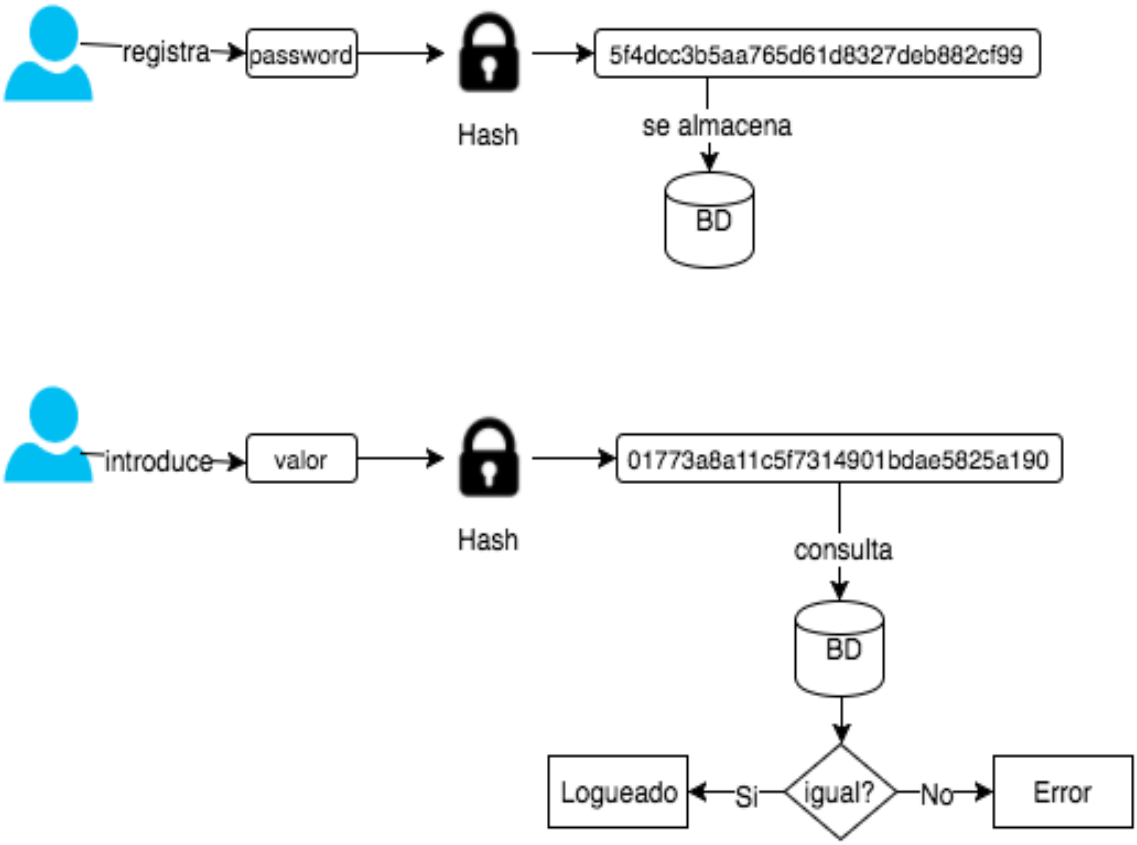


**Ilustración 4. Esquema funcionamiento hash**

Además de la seguridad en la base de datos, la aplicación también debe tener seguridad propia **[11]**:

- Implementar de manera correcta las funciones recordar contraseña o recordar correo electrónico. No facilitar a los atacantes el poder obtener la contraseña en plano de los usuarios, o el poder insertar ellos un correo para recibir los pasos.
- No mostrar mensajes de error con mas datos de los necesarios, como puede ser indicar si se ha escrito mal el usuario o la contraseña, por separado, al existir un fallo en

la autenticación; o la muestra de trazas de error al mostrar datos de la base de datos.

- Implementar de manera correcta la inserción de datos. Evitando la inyección de SQL, el *Cross-Site Scripting,* o el desbordamiento del buffer.

- Evitar la interferencia de datos. Un usuario solo podrá visualizar los datos que le pertenezcan, y no poder ver que tienen otros usuarios dentro de la aplicación.

- Correcto uso de las sesiones de usuario, tanto de la sesión en si como de las variables que esta pueda contener.

- Comprobación en servidor de todos los datos insertados por el usuario dentro de la aplicación, para que no puedan ser modificados.

Por último, este front-end, debería comprobar que los datos que se introducen mediante los sensores no están trucados por el usuario, tanto de forma consciente como inconsciente, como pudiera ser el estar sentado con el móvil cerca de una calefacción, lo que provocaría que los datos de la temperatura no fueran reales. Por ello, se debería comprobar que estos datos siguen una correlación con los anteriores datos y que los valores se encuentran dentro de un rango correcto.

# 4. Implementación

## *4.1 Introducción*

En este capítulo se tratará la implementación final realizada dentro del diseño realizado de forma inicial. Este prototipo de front-end, se ha querido realizar para dar una funcionalidad básica, con diversos módulos que constan del tratamiento de los logs, y de la información y registro de los dispositivos de la red. Además se ha implementado la autenticación para los usuarios junto con una base de datos que almacena diversa información tanto de usuarios como de dispositivos.

Junto a esto, se ha realizado un diseño, en formato de aplicación web, que permite al usuario poder interactuar con los datos mostrados, por ejemplo filtrando las tablas resultantes. Además se dispondrá de un panel de control en el cual de forma muy sencilla se podrá seleccionar cual de las opciones de la aplicación se desea consultar.

## *4.2 Framework usado para implementar el diseño*

Dadas las necesidades del diseño, en la implementación se ha usado Vaadin, como framework para programar  la aplicación web.

Este framework, permite escribir toda una aplicación web, con Java, un lenguaje orientado a objetos, característica muy útil sobre todo a la hora del diseño, puesto que no hay más que crear los objetos deseados y colocarlos,

modificando sus propiedades, en los lugares que deseamos. Además, con los listeners de cada objeto se puede controlar las acciones que el usuario realiza sobre dichos objetos, como puede ser un click sobre un botón o la entrada de un valor en el campo para filtrar. Con la lógica básica de Java, se puede programar el comportamiento que la aplicación debe seguir cuando los listeners detecten alguna acción por parte del usuario cambiando el contenido de la aplicación con mucha facilidad (mostrando notificaciones, cambiando contenido de textos, o tablas o redirigiendo a otras páginas).

Este framework, creado hace no más de 5 años, se puede usar dentro de los IDEs mas extendidos, como puede ser Eclipse o NetBeans, dada la arquitectura del mismo, provoca que el usuario no tenga que descender a un nivel muy bajo para programar la aplicación, el es quien convierte el código escrito en Java, para que sea legible por los navegadores y pueda ser mostrado por pantalla.

El diseño de la aplicación, se puede realizar de forma responsiva, es decir, se adapta a los distintos formatos de pantalla de dispositivos, sea una tableta, un móvil o un ordenador personal, característica muy útil con la diversidad de dispositivos usados en la actualidad.

## *4.3 Descripción de los módulos*

El prototipo realizado consta de 2 módulos que son los que realizan con conexiones con el *Clearing House*. Estas conexiones, debido a pruebas

realizadas sobre la central la cual se sobrecargaba si se solicitaban datos continuamente, son realizadas cada cierto tiempo, siendo guardado un valor temporal en la base de datos( de la cual se hablará en siguientes apartados) que es renovado con cada información nueva que el *Clearing House* envía al front-end. Los dos módulos implementados, reciben información de forma automática por parte de la central sin tener que realizar el usuario ninguna consulta por su parte, nada mas allá que elegir en el panel de control cual de las opciones desea visualizar. Ambos módulos son independientes entre sí, y realizan la información de forma autónoma, siendo tratados por dos caminos diferentes, las peticiones que lleguen al front-end, serán comprobadas por Spring, quien diferenciará si son alguno de estos módulos o por contra son peticiones del usuario de navegación, en caso de esta ultima opción, la petición es traspasada a Vaadin y recogida por el servlet que este framework tiene. Si es de estos módulos, Spring tratara estas peticiones tal y como se explicará en los siguientes apartados. Las peticiones en las que el Clearing House envíe datos o información a la aplicación, se realizarán mediante el método *POST* de HTTP.

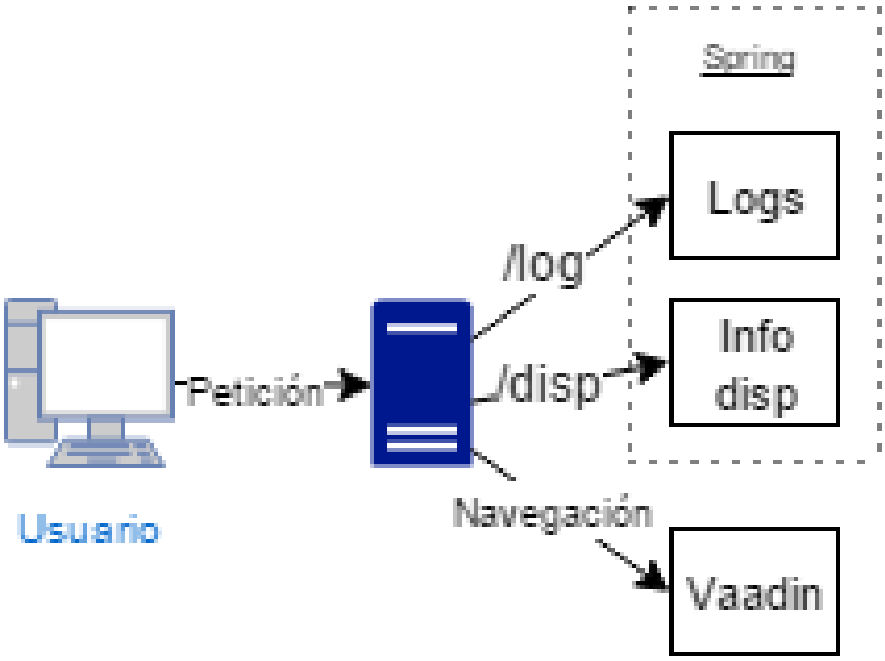


**Ilustración 5. Esquema funcionamiento recepción peticiones**

En el anterior capítulo de esta memoria, se explicó el diseño que debería tener este front-end, con todas las características necesarias para poder explotar al 100% la plataforma en la que se va a integrar.

Finalmente, este proyecto, debido a que se realiza una versión incial de este diseño, se ha decidido realizar un prototipo, con una funcionalidad básica para que el usuario pueda probar el funcionamiento de la aplicación.

Esta funcionalidad básica consta de dos módulos, que se detallaran en los siguientes apartados, además de una autenticación básica, todo ello junto con una base de datos que guarde la información del usuario y de los dispositivos.

Dado que en el diseño original, el usuario podía desarrollar los experimentos y subirlos a la plataforma, para luego consultar el estado de los mismos, se ha

mantenido, en parte, esta funcionalidad, y aunque el usuario no puede subir los experimentos, si puede consultar y filtrar los dispositivos que existen además de sus características. Los usuarios tendrán asignados unos experimentos en la base de datos y de los que también podrá consultar los logs, funcionalidad que si se ha implementado en este prototipo.

En este prototipo, no se ha implementado la funcionalidad diseñada en la que el usuario tenía una cuota la cual era limitada, y podia aumentar dependiendo de distintos factores, dicha cuota, en este prototipo es ilimitada, debido a la funcionalidad básica que se ha querido crear con esta implementación de parte del diseño.

## *4.4 Módulo de logs*

Este módulo será el encargado de recibir del Clearing House los logs que este enviará y los almacenará en la base de datos a la que el front-end está conectado.

Se ha configurado Spring, en el archivo web.xml para indicar que url será a la que el Clearing House enviará los logs para el front-end. Se configura con la siguiente instrucción:

```
    <servlet>
 <servlet-name>Logs</servlet-name>
<servlet-class>
org.springframework.web.servlet.DispatcherServlet
      </servlet-class>
```

```
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>Logs</servlet-name>
    <url-pattern>/log</url-pattern>
  </servlet-mapping>
```

Una vez se tiene mapeado la peticiones, se indica la existencia de dicho mapeo en el *WEB-INF.*

Esta configuración se realiza para separar las peticiones que llegarán al front-end, en este caso tramitaremos las que lleguen bajo la URL " /log/ ", tras esto, se ha implementado un manejador, o controlador, de dichas peticiones que lleguen con esa URL.

En este controlador, se ha implementado la lógica que la aplicación realizará tras recibir el Json que porta la información del log y el log en si mismo.

El formato de los Json relativos a logs, es el siguiente:

```
{msgId: 12,
sender: {name: "clearing  house", nodeId: "ch_id", AEId:"dsds"
logEntries:
[{expId: "3243423",
AEId: "23423423",
Date:"05/24/2016 1:27",
message: "Application::method:: Cannot load initiate sensor",
attachments: [ "fn84fn84fn84---base64String",
"fn84fnrewrewrew84fn84---base64String"]},
{expId: "32344223",
AEId: "12332423",
Date:"05/24/2016 1:12",
message: "Application::method:: Hello world",
attachments: [ "ffsdfn84fn84fn84---base64String",
"fn84fnrewrewrew84sdfwefefewffn84---base64String"]}
}
```

Como vemos, el Json tiene un campo expId, el cual indica cual es el Id del experimento, que será luego usado para mostrar los experimentos al usuario que este usando la aplicación.

Para tramitar estos Json, se ha usado la librería Gson de Google, con la cual, se puede leer este formato de archivos, de forma muy sencilla.

Para usar esta librería, se ha creado una clase de Java, con los mismos atributos que propiedades tiene el Json, todos estos atributos, contarán con métodos *get* y *set* para obtener o definir a los mismos.

Una vez se procesa la petición, se extrae el Json, y del mismo se saca la información simplemente con línea de código, gracias a Gson:

```
Log logObjeto = gson.fromJson(log_recibido, Log.class);
```

Siendo " log_recibido " el buffer leído de la petición. Tras esto, ya se tiene el log, en un objeto Java, con el cual podemos interactuar como si de cualquier otro objeto se tratara.

Ahora, nos queda guardar este log en la base de datos, insertando los datos relativos al id del experimento y la fecha. Con los métodos mencionados anteriormente, esta operación es muy sencilla, no hay mas que obtener el valor del atributo *expId*, *date* y *message* que tendrá el objeto *logObjeto* e insertarlo en la base de datos, en las columnas relativas.

La inserción en la base de datos, se realizará mediante el uso de JPA, tal y como se comentará en los siguientes apartados. Dado que se tiene guardado el objeto lobObjeto y JPA se basa en el uso de clases llamadas POJOS, las cuales envuelven el elemento y lo insertan en la base de datos.

Estos logs, se mostrarán al usuario cuando este acceda a la opción *Logs* dentro del panel de control de la aplicación. Debido a que se indica cual es el Id del experimento, el usuario podrá elegir de cual los experimentos quiere ver el log y así poder observar cual de los experimentos está funcionando correctamente y cual no.

## *4.5 Módulo de información y valores*

Este módulo no tiene grandes diferencias con el anterior, en este caso, se recibirá los valores de los sensores que ya están en la base de datos y la información relativa a los nuevos que se quieran agregar.

Realizaremos un mapeo de las peticiones que se vayan a recibir para diferenciar de las que van para *Vaadin* o para el módulo anterior de las que son para este módulo.

```
    <servlet>
 <servlet-name>Dispositivos</servlet-name>
    <servlet-class>
org.springframework.web.servlet.DispatcherServlet
     </servlet-class>
     <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
     <servlet-name>Dispositivos</servlet-name>
     <url-pattern>/disp</url-pattern>
  </servlet-mapping>
```

Este módulo, como se ha explicado, consta de dos opciones, una para los valores y otra para los nuevos dispositivos, por ello, deberemos controlar de distinta manera estas peticiones para poder tramitar los JSON recibidos. De nuevo, mediante la URL de la petición, podremos saber cual es la intención del *Clearing House.*

Con la implementación de una clase con diferentes notaciones, podemos indicar a la aplicación que esta clase es un controlador (@controller) y diferenciar entre las distintas peticiones (@RequestMapping)

```
@Controller

public class DispositivoControlador {

@RequestMapping(value = "/valor", method = RequestMethod.POST)

   public void lecturaJSON () {
```

Con ello, cuando se reciba una petición *POST* con la url /disp/valor/, se tramitará dicha petición leyendo el JSON de igual manera que se realizó en el módulo 1, teniendo clases POJO's con los atributos que tendrán los experimentos que se recibirán, guardando su id, y sus valores.
Estos objetos, se almacenarán mediante JPA en la base de datos en la tabla *lecturas.* Para poder ser luego mostrada la información de los valores al usuario.

Para el caso que el JSON que se reciba sea con información de un nuevo dispositivo, se ha añadido en la misma clase que la anterior, un nuevo mapeo con la notación (@RequestMapping) para procesar las peticiones relacionadas con este tema.

```
@RequestMapping(value = "/nuevoDisp", method = RequestMethod.POST)

   public void lecturaJSON () {
```

Ahora cuando se reciba una peticion POST con la url /disp/nuevoDisp, se leerá el JSON con la información del nuevo dispositivo para que sea añadido a la base de datos de la misma manera que se ha hecho en con los valores.

Los JSON relativos a estos nuevos dispositivos, tienen la siguiente forma:

```
{msgId: 23,
sender: {name: "clearing  house", nodeId: "ch_id", AEId:"ch_app_id_at_ch_id"
devices:
[{dispId: "3243423",
Date:"05/24/2016 1:27",
Capabilities:
[
{ friendlyName: "Temperature sensor",
  description: "http://location_of_description.net/sensor.xml,
  descriptionType: " http://www.opengis.net/sensorml/2.0"
},
{ friendlyName: "Actuator",
  description: "http://location_of_description2.net/actuator.xml,
  descriptionType: " http://whateverOtherIdentifierAsUri.net"
}
]}]
```

Estos dispositivos, se añadirán a la base de datos, y serán mostrados al usuario que podrá filtrar por características, por ejemplo en este caso, este dispositivo tiene sensor de temperatura y un actuador. El usuario, dado que es un prototipo, no podrá seleccionar estos dispositivos, solo visualizarlos dentro de la opción del panel de control llamada Dispositivos.

## *4.6 Autenticación*

Dentro de las distintas opciones para el login de los usuarios que se diseñaron de forma original, se ha implementado la autenticación básica, en la que el usuario que desea usar la aplicación, debe insertar un nombre y una

contraseña. Estos datos están almacenados en la base de datos a la cual consulta la aplicación para comprobar que los datos introducidos son correctos.

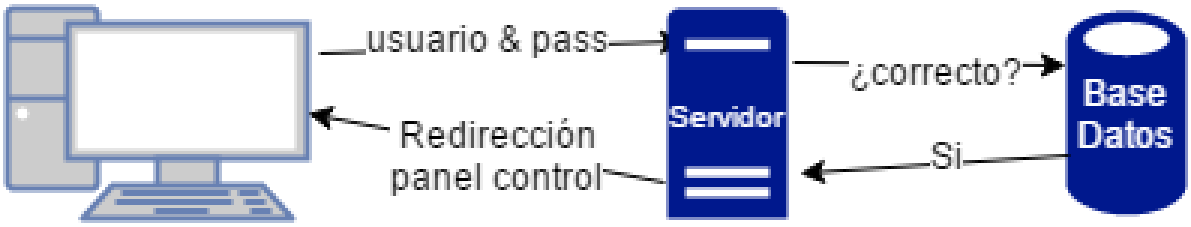


**Ilustración 6. Diagrama autenticación correcta**

En caso de que no lo sean, se le vuelve a mostrar la misma página, mostrando, ademas, un pequeño mensaje en el que indica que introdujo mal los datos, si por contra, el usuario introdujo información que es correcta, se le redirige a la página que muestra el panel de control con las opciones elegibles por el usuario.

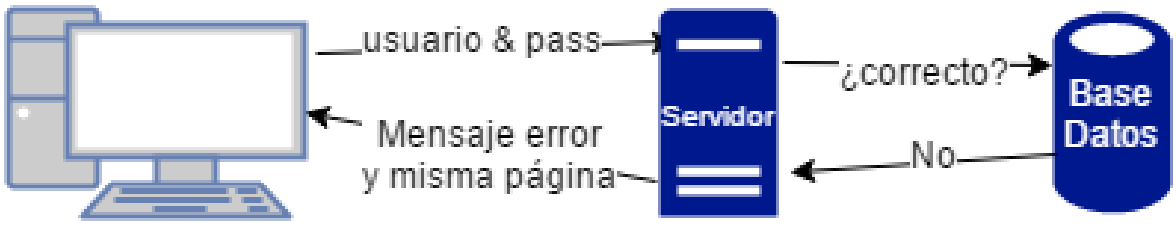


**Ilustración 7. Diagrama autenticación incorrecta**

El programa pide un correo electrónico, que identifica al usuario y una contraseña. Ambos datos se envían a la aplicación cuando el usuario hace click en “enviar”. Si la autenticación ha sido correcta, se almacena en la sesión un

parámetro que identifica al usuario para poder hacer las consultas a la base de datos y mostrar y la información relativa a ese usuario y no a otro. Este parámetro se mantiene durante todo el tiempo que la sesión dure y solo es borrado cuando el usuario seleccione el botón de “Salir” que dispondrá durante toda la aplicación.


**Ilustración 8. Autenticación de la aplicación**

## *4.7 Base de datos*

En el diseño, se relataron las características que debería cumplir la base de datos para el diseño de este front-end. Dentro de las posibles opciones que existen actualmente, habiendo una gran diversidad de posibilidades, se implementó esta parte del sistema con una base de datos SQLite, basada en SQL, como su nombre indica, que dado el pequeño tamaño de su biblioteca y

su bajo tiempo de respuesta a las peticiones que se realizan, es idónea para el objetivo deseado.

Dentro de la base de datos, se crearon varias tablas:

- Usuarios

  Esta tabla contiene la información relativa todos los usuarios de la aplicación. Está formada por 7 columnas con los datos de referencia del usuario, como puede ser el correo, su nombre, el de su empresa y el número de experimentos que actualmente tiene en la plataforma. Además de esa información que es de referencia, se almacena el id que identifica al usuario en la aplicación y su nombre de usuario y contraseña, que es lo que se usa para comprobar la autenticación a la hora de entrar en la aplicación. A esta tabla se le ha asignado el nombre: “usuarios”.

- Experimentos

  Esta tabla contiene todos los experimentos existentes en la plataforma indicando quien es su creador, cuantos dispositivos tiene dicho experimento y una lista con los id’s de los dispositivos que participan de dicho experimento además del nombre que el usuario puso a este experimento.

  Esta lista, se inserta como valor en una de las columnas que tiene como tipo de datos varchar, separando cada id con una coma. Como sabemos cuál es el número de dispositivos, pues hay una columna que indica este valor, leer esta lista de id’s no es nada complicado.

A esta tabla se le ha llamado “experimentos”

- Dispositivos

  Esta tabla contiene todos los dispositivos que existen en la red, mostrando su nombre: “friendly_name”, su descripcion y el tipo de dicha descripción, estos dos ultimos son url que indican como son estos dispositivos. Todos estos datos se reciben del Clearing House y se insertan en la base de datos. Tienen un id que los identifica dentro de los experimentos .

  A esta tabla se le ha llamado “dispositivos”

- Logs

  Esta tabla es la que indica a la aplicación, en que ruta del servidor se ha guardado los logs que el clearing house ha enviado indicando a que usuario pertenece y la fecha en la que fue insertado en la base de datos.

  Para ello, se han creado 5 columnas, 3 de ellas dan información de los logs (id_log, fecha, log), otra columna indica a que experimento esta referido y una última que indica cual es el id del usuario al que pertenece dicho experimento. Estos logs, muestran al usuario toda la información de los procesos que se han ido haciendo en sus experimentos, por ejemplo si un sensor esta caído o si alguno de sus experimentos ha dejado de funcionar. Los logs son guardados tal cual como un varchar  en la base de datos y son leidos para

mostrarlos al usuario. El “id_exp” indica de qué experimento es el log de esa fila.

Tabla con nombre “logs”.

- Lecturas

  Esta tabla contendrá los valores leídos por los sensores y recogidos por el Clearing House. Tendrá 3 columnas, una con el id del experimento, otra con el id del dispositivo y una ultima con el valor en si, esta última siendo de tipo varchar.

  Esto hará, que cada lectura de un dispositivo, tendrá una entrada en la base de datos y cuando mas tarde, se quiera mostrar esa información al usuario, solo se tendrá que buscar cual es el id de su experimento y en caso de querer filtrar por id del dispositivo, se tendrá un valor para ello.

  La tabla se ha creado con “lecturas”, como nombre

## *4.8 Navegación de la aplicación web*

Vaadin proporciona dos opciones posibles para la realización de una aplicación web: la primera es el uso de una sola página, que se va recargando dependiendo de como interactúe el usuario con la misma; y una segunda, en la que el usuario va siendo re-dirigido a diversas url’s en la que la información que desea ver es mostrada.

En este proyecto, se ha optado por la segunda opción, una navegación por URL's en la que se ha creado una vista distinta para cada una de las opciones del panel de control. Estas vistas, han sido creadas con un Add-On que el framework permite, llamado Vaadin Designer. Este Add-On, permite la selección dentro de una gran variedad de componentes que componen la paleta, y que mediante el arrastre al lienzo, es posible colocarlos dentro de la pantalla. Estos componentes, tienen unas propiedades que pueden ser modificadas, como por ejemplo su nombre, el valor que muestran en caso de ser un texto o la anchura y altura que ocupan dentro del lienzo. Vaadin autogenera un documento html a la par que un documento .java. El primero es el que los navegadores leen para poder mostrar los componentes en pantalla y el segundo es realmente una clase interfaz de Java, que se ha de implementar más tarde para poder crear la lógica de negocio de dicha página web.

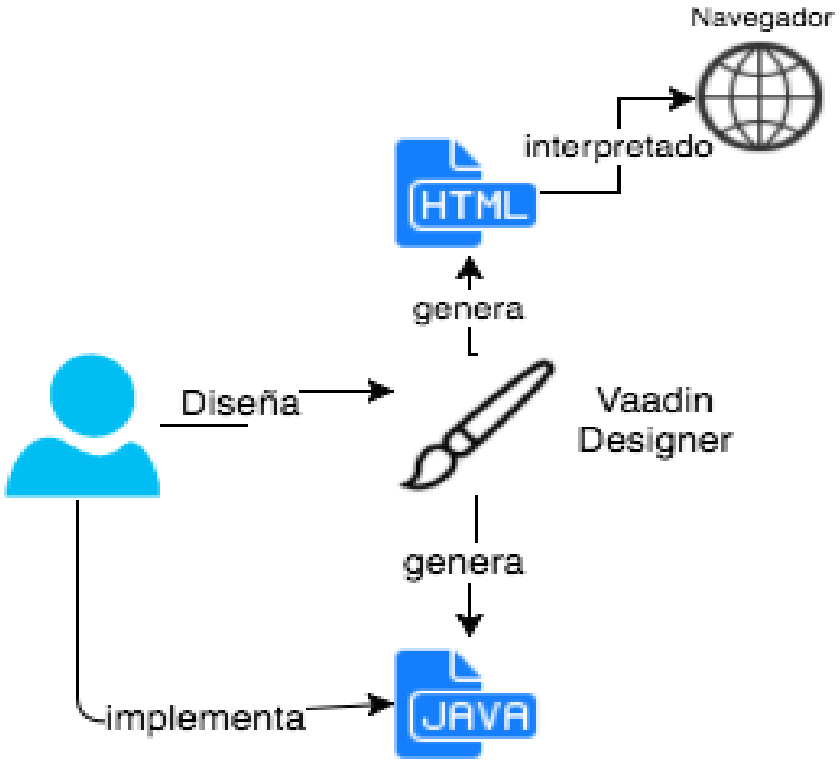


**Ilustración 9. Diagrama funcionamiento Vaadin Designer**

En esta aplicación se han creado 6 vistas:

1. Panel de control. Es la vista que muestra al usuario las opciones posibles que puede seleccionar para ver. Cuando el usuario pincha en una de ellas, se le redirige a la página que desea. Cada botón tiene un listener, que se activa cuando se hace click y que no hace mas que redirigir.

2. Experimentos. Esta vista muestra todos los experimentos que el usuario que está en sesión tiene actualmente creados.

3. Dispositivos. En este prototipo, esta vista solo muestra todos los dispositivos que existen en la red, para poder saber que tipo de sensores tienen y así poder saber que experimentos podrían ser creados.

4. Información del usuario. Muestra la información personal del usuario que esta visualizando la aplicación web.

5. Logs. Muestra todos los logs de los experimentos que el usuario de la sesión tiene dentro de la plataforma.

6. Inicio de sesión. Es la página inicial que se muestra cada vez que un usuario accede a la aplicación, no muestra mas que un campo para insertar el correo y otro para la contraseña, junto con un botón para enviar dicha información al servidor.

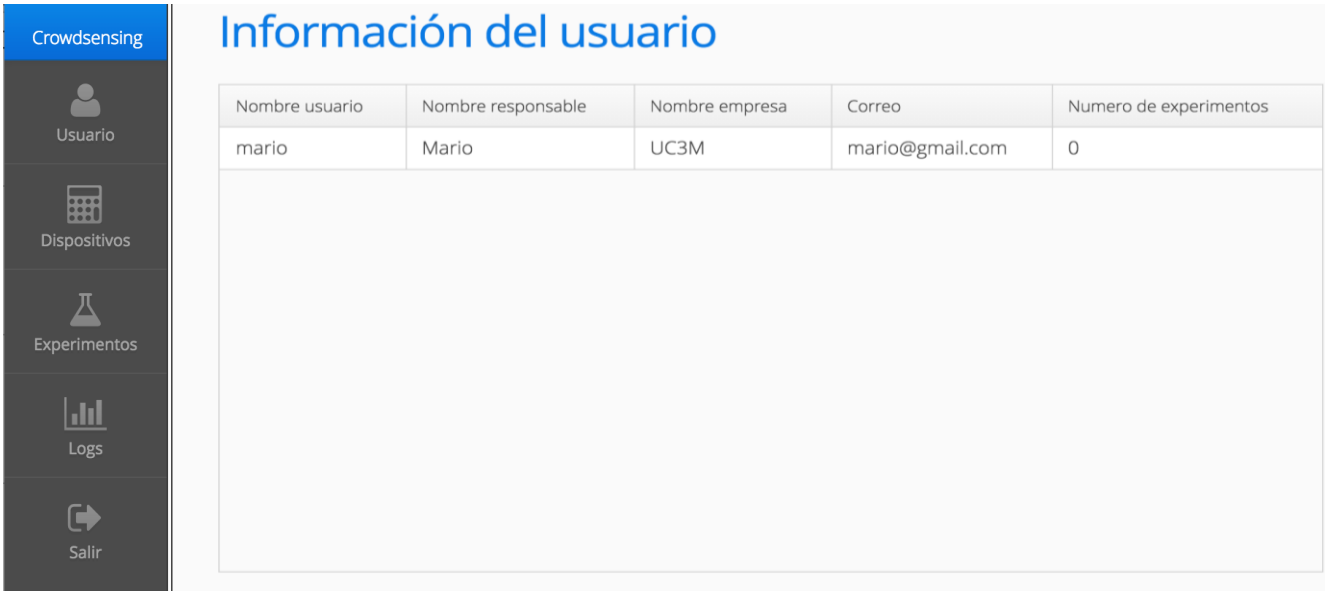


**Ilustración 10. Muestra información del usuario**

Todas las vistas, excepto el panel de control comparten un menú lateral que permite moverse a las distintas opciones del panel de control sin tener que pasar previamente por él. Para cerrar sesión, estas vistas también disponen de un botón, que elimina el parámetro guardado de la sesión y cierra por completo la misma.

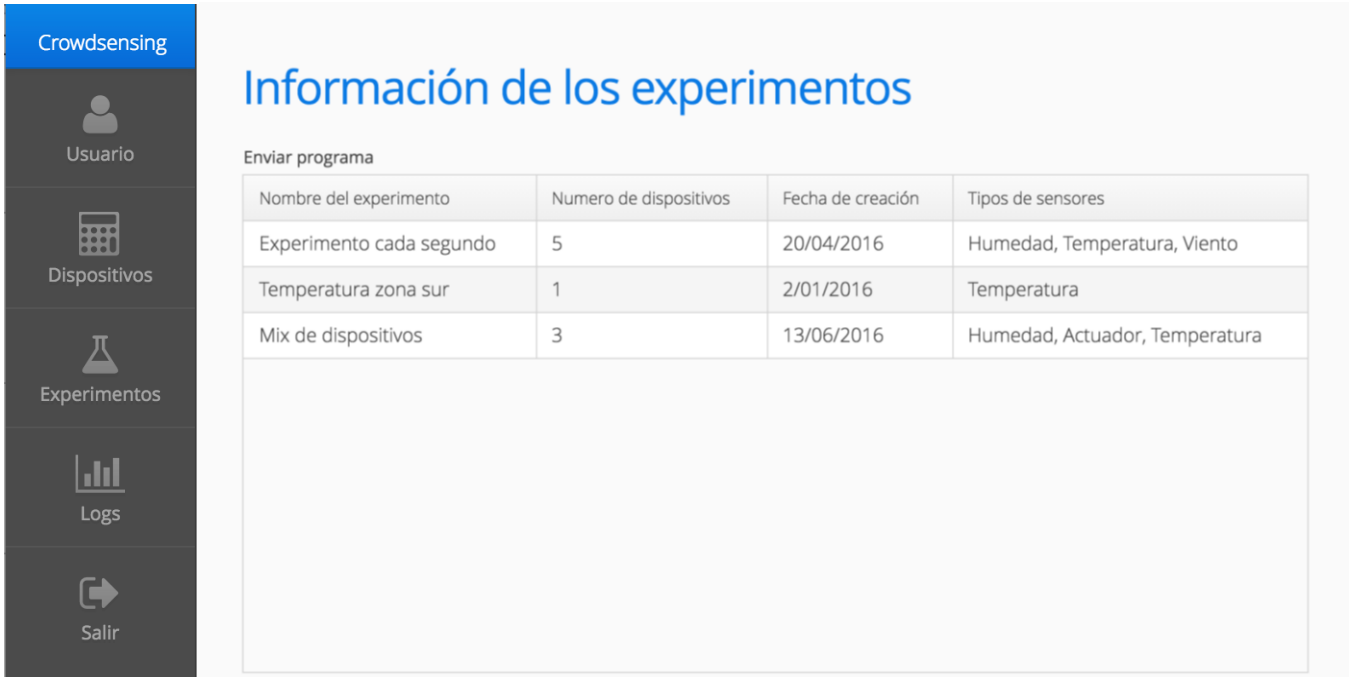


**Ilustración 11 Muestra información de los experimentos**

Dado que el usuario, desea saber si existen dispositivos con sensores específicos para poder crear sus experimentos, se he implementado un filtro en las tablas que muestran los dispositivos disponibles. Estas tablas filtradas, constan de campos en la parte superior de las mismas, en los cuales el usuario puede ingresar letras o palabras, para que solo se muestren en las tablas los datos que coincidan con los valores que el usuario ha ingresado. Con Java 8, y el uso de streams, es muy fácil el uso de estos filtros, pues se crea un stream (un flujo) sobre el cual se realiza una operación, como puede ser que su campo nombre comience por “A”, y los valores que no coincidan con este filtro, serán

eliminados del stream, por ello, nos quedará un flujo nuevo, con los datos que coinciden con el filtro que el usuario ha querido aplicar.

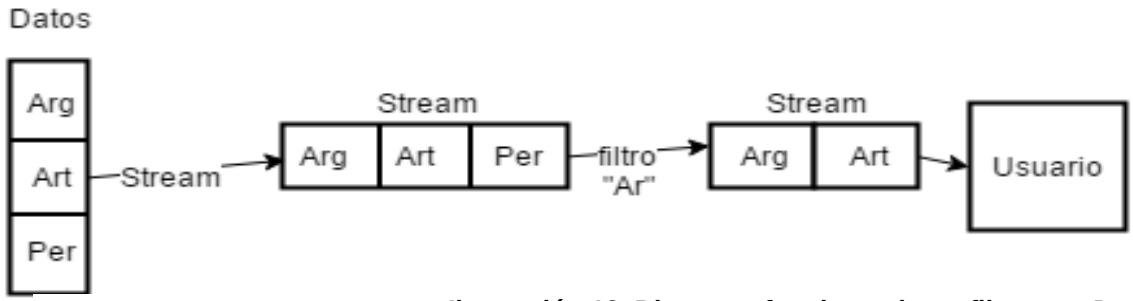


**Ilustración 12. Diagrama funcionamiento filtros en Java 8**

## *4.9 Comunicación con la base de datos*

Como se ha visto, se realizan llamadas a la base de datos desde la aplicación, tanto para mostrar información al usuario como para almacenar nueva información que es recibida. Por ello, se ha decidido usar la API de Sun Microsystems conocida como JPA, implementada para Java, el lenguaje que Vaadin usa por lo cual el lenguaje usado en esta aplicación.

En el proyecto, se han creado entidades, es decir clases de Java, que persisten la información asociada a una tabla de datos, por ejemplo los usuarios, son almacenados en una clase con diversas propiedades (nombre, numero de experimentos, etc), que son luego ingresados en la base de datos. Estas clases son identificadas como entidades con la instrucción “@entity” al inicio de la misma, con esta instrucción se informa de la persistencia de dicha clase. En este caso, en las entidades se ha informado a la base de datos mediante notaciones cual son el tipo de valores de las distintas tablas de la base de datos.

```
@Entity
public class Usuario {

        @Id
        @GeneratedValue(strategy = GenerationType.AUTO)
        private Long id;
        @NotNull
         @Size(min = 2, max = 24)
        private String password;
        @NotNull
        @Size(min = 2, max = 24)
        private String nom_usuario;
        @NotNull
        @Size(min = 2, max = 24)
        private String nom_resp;
        @NotNull
        @Size(min = 2, max = 24)
        private String nom_emp;
        @NotNull
        private String correo;
        private Long num_exp;
```

De la misma manera, se puede realizar la función inversa, es decir, no solo se pueden crear objetos que implementan las entidades para insertar la

información de dichos objetos en la base de datos, sino que también se puede obtener esta información y crear objetos con la misma para así poder ser representado en pantalla por ejemplo.

Con Vaadin, incluso la conexión con la base de datos no es necesaria realizarla de forma manual, creando un EntityManager, existe un Add-On de este framework, con el cual se crean contenedores de las entidades que se han implementado.

Con la siguiente instrucción se llena un contenedor de usuarios con los contenidos en la base de datos debido a que al método “make” se le indica mediante sus parámetros que estamos obteniendo objetos de la entidad Usuario, en este caso:

```
JPAContainer<Usuario> usuario_entidad;

usuario_entidad =
JPAContainerFactory.make(Usuario.class,Inicio.PERSISTENCE_UNIT);
```

En caso de querer solo alguno de estos usuarios, por ejemplo en si se desea comprobar la autenticación, se pueden aplicar filtros a dicha consulta en la base de datos, lo que equivaldría a una sentencia SQL con un “WHERE”. En Vaadin, esto se realiza con la siguiente instrucción:

```
usuario_entidad.addContainerFilter(new Compare.Equal("correo", correo));
```

Siendo el primer parámetro el valor de la columna que se desea consultar en la base de datos, y el segundo parámetro, en este caso un String con el valor del correo para comprobar si existe algún usuario con ese correo.

Para los dos módulos que reciben información del Clearing house, como se ha explicado en apartados anteriores, se ha implementado con Spring, por ello no se puede usar Vaadin para encapsular los datos como se ha hecho para el resto.

Debido a que en algunas de las tablas creadas, la columna que relaciona una con otra es la de su “id”, se ha creado una lógica en la aplicación para poder obtener, por ejemplo quien es el usuario de un experimento, leyendo el valor en la tabla “lecturas” uniendo con la tabla usuarios y experimentos, sabiendo con esta ultima quien es el usuario al que pertenece un experimento, y dando nombre a este usuario con la primera de las tablas mencionadas.

Para poder persistir la información de estos módulos en la base de datos, se ha usado, en este caso, DAO (del inglés Data Object Access) complementando a Spring, que hemos usado para poder tramitar las distintas peticiones que recibiremos.
Se ha creado, en primera instancia una capa de DAO, creando una clase abstracta que será implementada e incluso extendida, esta clase, tiene los prototipos de métodos para realizar las distintas acciones en la base de datos, como puede ser: buscar uno, varios o todos los elementos, crear o actualizarlos e incluso eliminarlos. Esta clase se implementará para los distintos elementos que se desean obtener, como puede ser los experimentos, las lecturas o los logs, con sus respectivas clases POJO.

Por otra parte, se ha configurado la persistencia de JPA, en una clase, donde se establecerá el *entitymanager*, que es un objeto que se crea para establecer

la conexión con la base de datos para que después la clase explicada mas arriba pueda establecer las sentencias que sean oportunas.

Además de todo esto se ha de modificar el xml, indicando toda la información relativa a la base de datos, junto con cual es el *entitymanager* que generará las conexiones con la base de datos.

# 5. Pruebas

En este capítulo se detallaran las pruebas realizadas en el desarrollo de este front-end y los resultados obtenidos en estas. Estas pruebas han sido realizadas tanto tras la implementación del prototipo, para comprobar el correcto funcionamiento del mismo, como con anterioridad, para poder comprobar distintos parámetros de la plataforma para elegir que opción de las barajadas, que cumplen con el diseño, sería mejor implementar

## *5.1 Pruebas de carga sobre el Clearing House*

Como se habló en el diseño original, el front-end, debía comunicarse con el *Clearing House*, la central que inter-conecta toda la plataforma. Como idea inicial, se pensó en solicitar los datos a esta central cada vez que los usuarios los requiriesen, es decir, si un usuario quería saber que valores tenía sus experimentos, se hacia una petición al Clearing House y este respondía. Además, y como se sabe, esta plataforma tiene como novedad, la posibilidad de configurar los tiempos de obtención de estos datos, por ello, cada experimento, obtendría valores en distintos períodos de tiempo, de nuevo provocando que el *Clearing House*, recibiera y enviara peticiones.

Dado que se vio que quizás el *Clearing House* iba a recibir una carga elevada de tráfico. Se quiso comprobar que fuera capaz de soportar dicha carga, para poder realizar esta idea en el front-end. Estas pruebas revelaron que el

Clearing House no era capaz de soportar tanta carga como la que podría producirse a la hora de crear mas de 3 ó 4 experimentos. Por ello, se decidió realizar la implementación relatada en capítulos anteriores. Los datos son consultados en el *Clearing House* de forma periódica y automática y son almacenados en la base de datos de la que el front-end dispone, para poder luego ser servida a la aplicación cuando el usuario la muestre.

Por otra parte, y aunque no formase parte de este front-end como tal, pero si de la plataforma, se decidió cambiar la forma en la que los dispositivos enviaban los datos al *Clearing House,* tomándose los datos en los períodos que el usuario había configurado, pero no siendo enviados en ese instante, sino siendo acumulados durante un período de tiempo para poder enviar varios a la vez para así rebajar la carga de peticiones que se recibirían.

## *5.2 Pruebas de funcionamiento de la aplicación*

Una vez se terminó de implementar el prototipo del front-end. Se procedió a comprobar el correcto funcionamiento del mismo, para comprobar que se cumplen las especificaciones que se habían relatado. Se realizaron diversas pruebas para corroborarlo:

- ✓ Prueba de login. Se ha comprobado el correcto funcionamiento de la autenticación que da acceso a la aplicación únicamente en el caso de que se introduzca de forma correcta el correo electrónico y la contraseña. En caso de que no sean correcto alguno de estos datos, la aplicación

informa solo de que alguno de las dos datos se han introducido de forma errónea, cumpliendo así con las especificaciones de seguridad, y redirigiendo a la misma página para volver a introducir los datos. El cierre de sesión mediante el botón que se muestra en el menú lateral, también funciona de forma correcta, cerrando la sesión, y evitando que el usuario acceda de nuevo, si no ingresa su usuario y contraseña.

- ✓ Prueba de muestra de información. La aplicación, solo debe mostrar los datos referidos al usuario que ha iniciado sesión, este hecho, se ha comprobado intentando acceder a datos de otros usuarios, y se ha comprobado que el funcionamiento es correcto, y solo se muestran los datos de ese usuario.

- ✓ Prueba de filtros. Dado que los usuarios deben poder filtrar los dispositivos según sus características, se ha comprobado que los filtros funcionen de forma correcta, mostrando solo los dispositivos que coincidan con aquellos filtros que el usuario introduzca.

Las tres pruebas fueron superadas, por lo que la aplicación tiene un comportamiento correcto en base al diseño que se realizó en anteriores capítulos de este proyecto.

# 6. Planificación y entorno socio-económico

## *6.1 Planificación*

En este capítulo se explicará cual ha sido la planificación del proyecto detallando los días empleados para las distintas partes y las dependencias que tenían entre ellas.

Esta planificación se ha desarrollado determinando unas 2 horas y media de trabajo al día de media, subiendo en época de menos carga de trabajo y bajando en época en la que la carga externa a la realización del trabajo era mayor. Se comenzó a trabajar en el mismo a inicios del año 2016, finalizando la redacción de esta memoria a mediados del mes de Junio.

Se ha dividido el proyecto en 5 tareas, las cuales se realizaron de forma consecutiva, dependiendo cada una de la finalización de la anterior.

| Num | Tarea | Duración (días) | Fecha inicio | Fecha final | Tarea Precedente |
|---|---|---|---|---|---|
| 1 | Recogida Información | 25 | 01/02/2016 | 04/03/2016 | - |
| 2 | Diseño | 21 | 07/03/2016 | 04/04/2016 | 1 |
| 3 | Implementación | 30 | 05/04/2016 | 13/05/2016 | 2 |
| 4 | Pruebas | 6 | 16/05/2016 | 23/05/2016 | 3 |
| 5 | Redacción memoria | 16 | 24/05/2016 | 17/06/2016 | 4 |

**Tabla 1. Tareas planificadas**

Para mostrar de forma más gráfica el progreso de tareas a lo largo del tiempo, se ha realizado un diagrama de Gantt, se debe tener en cuenta que las fechas tanto del diagrama como de la tabla, son orientativas.

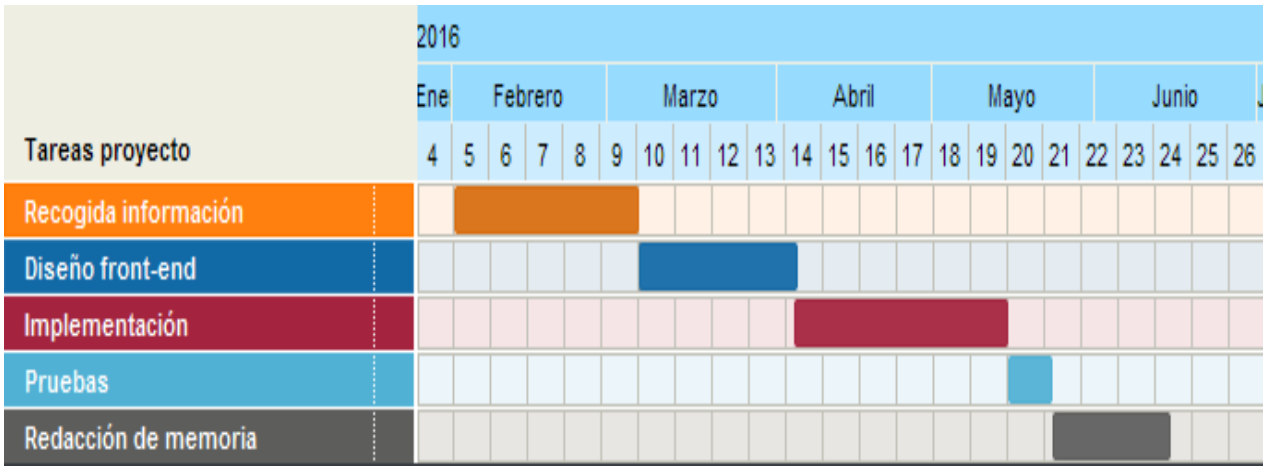


**Ilustración 13. Diagrama de Gantt**

## *6.2 Entorno socio-económico*

Como es bien sabido, desde hace algo menos de 10 años, la economía mundial se encuentra en una gran depresión, los distintos mercados de productos han visto como disminuían sus beneficios, incluso acusando grandes pérdidas. El mercado de los dispositivos móviles aun con una reducción en sus beneficios, ha visto como aumentaba de forma exponencial la venta de dispositivos móviles, con una penetración de mercado que en 2020, se calcula será del 75%. Estos dispositivos móviles, cada vez son más inteligentes y mas completos, con grandes procesadores, memoria y sensores de alta precisión.

Estos móviles disponen, casi al 100%, de una tienda de aplicaciones en la que los usuarios pueden elegir entre cientos de miles de aplicaciones con diversas funciones y objetivos.

El uso de estos dispositivos, no se restringe a una banda de edad en especial, la penetración en el mercado, ha llegado a todas las franjas de edades, y una aplicación de mensajería como *Whatsapp,* es usada tanto por personas de 70 años, como por niños que recién acaban de recibir su primer dispositivo. Por ello, es de vital importancia que estas aplicaciones que son publicadas para los distintos sistemas operativos, sean de gran sencillez y con una cuidada interfaz, que atraiga a los usuarios para usarlas.

En el caso de aplicaciones, o plataformas en la que los usuarios ceden información o datos, para crear una gran comunidad, como puede ser el ejemplo de Wikipedia, se está perdiendo la idea original del altruismo, el sentimiento de ceder estos datos para el bien común está siendo absorbido por las ansías monetarias de la sociedad actual, la cual necesita de un incentivo para realizar la gran mayoría de acciones de su día a día. Por ello, este tipo de plataformas, está empezando a incluir unas cuotas tanto de aportación como monetarias, para poder acceder a la información que alberga, en ocasiones de forma voluntaria como actualmente Wikipedia, que ha iniciado una campaña de donaciones; y en otras de forma necesaria para poder actuar a distintas funcionalidades.

Como resultado del aumento del uso de los dispositivos móviles unido a la fabricación de grandes sensores incluidos en estos y a la posibilidad de

obtener beneficios, las comunidades de Crowdsensing están aumentando en los últimos años, apostando, incluso, grandes marcas como IBM o Cisco, por este tipo de plataformas.

# 7. Conclusiones y líneas futuras de trabajo

## *7.1 Conclusiones*

El front-end que se ha implementado en este proyecto esta englobado en una plataforma de *Crowdsensing* ya desarrollada. En la actualidad, existen diversas plataformas con la misma finalidad: usuarios ceden el uso de sus sensores para formar una gran red en la que se puedan obtener diversos datos de diversos lugares, pero la novedad que tiene esta plataforma respecto a otras es la posibilidad de crear tus propios experimentos, en los que puedes seleccionar la configuración del proceso de obtención de los datos de estos sensores, como puede ser establecer el período de obtención de dichos datos o la localización de los mismos. Mientras que en la gran mayoría de plataformas existentes, estas características están predefinidas y son inamovibles. Debido a esta característica tan especial, la importancia de crear una aplicación usable y sencilla era de vital importancia, esta aplicación sería el nexo entre los usuarios y la plataforma, la forma en la que los usuarios pudieran ver toda su información personal y la de sus experimentos.

El objetivo de esta plataforma es que los usuarios puedan consultar los datos de los sensores que forman la red, siendo necesario que estos datos sean mostrados de forma clara y con un acceso fácil.

La ausencia de una estandarización en el mundo del Crowdsensing, dificultó en gran manera la implementación del diseño. Cada una de las plataformas

existentes usaban un formato y un envío distinto para los mensajes, factor indispensable para poder comunicarse con la aplicación. Por ello, se tuvo que diseñar cuál sería la forma en la que se enviarían los mensajes y cuál sería su formato, para poder luego ser tratados, finalmente se usó JSON mediante peticiones HTTP bajo el método POST, factor que facilitaba la lectura en el lenguaje Java.

La sencillez y facilidad de uso de la aplicación se consiguió con el uso de un Framework llamado *Vaadin* en el cual con el uso de Java como lenguaje de programación, se facilitaba de gran manera la creación de la lógica de la aplicación y por supuesto del diseño, con unas líneas claras e imágenes sencillas. Toda la información accesible en la aplicación, está a menos de 3 clics, evitando que el usuario tenga que buscar y buscar para poder encontrar aquello que desea consultar.

Toda la información del front-end, es guardada en una base de datos SQLite que persiste el modelo de datos que se diseñó de forma inicial. Esta base de datos tiene distintas tablas en las que se guarda tanto la información de los usuarios de la aplicación, como de los experimentos: valores, que dispositivos contiene, etc…Este tipo de base de datos, tiene una gran capacidad, y sobre todo una alta velocidad de acceso y procesamiento, lo que agiliza el uso de la aplicación.

## *7.2 Conclusions*

This project is a part of a existing Crowdsensing's platform. Nowadays, there are a few platforms with same goal: users yield his mobiles with sensors to

create a huge network where users can get different data of different locations. This platform has a special feature: users can create his own experiments to select where and when they want to extract this information. In others platforms, this features doesn't exist because it's common to all users. To use this element makes necessary the creation of a simple and intuitive application to connect platform with users. This users can visualize his own information and experiment's information, too.

This platform has a clear objective: users should can see the information of sensors existing in the network. This view of data should be simple and easy to access.

There's not any standardization in crowdsensing's platfoms, that unforeseen makes more dificult the creation of this front-end. Every platform used a different format of message and a different way to send this. So It was necessary to create a format of message and select how will be treat that messages. Finally, JSON was used with HTTP request with POST method. That aspect made the use of Java lenguaje easier.

Using a framework called Vaadin makes very easy to use the applcation. The front-end was developed with Java language that makes the logic of the application and design easier. The design has a clear lines with simple images. All information that users can see are in 3 clicks, no more. This feature avoid users have to spend a lot of time to find what they want.

The information of the application is saved in a SQLite's database that persist the data model that was design at the beginning. That database has a different

tables where user's information and app's information is saved. This information can be: device's information, data of sensors,…

That kind of database has a high access speed and a high processing capacity. That features improve the use of the application.

## *7.3 Líneas futuras de trabajo*

En este apartado, se explican cuales podrían ser las líneas de trabajo que se podrían continuar en el futuro continuando el trabajo que se ha realizado en este proyecto:

- Implementación de las funcionalidades que se diseñaron de forma original y en este proyecto no se decidieron realizar. Esto aumentaría las opciones de la aplicación que actualmente es un prototipo, consiguiendo que se pudieran realizar todas las funciones que se diseñaron.

- Persistir el modelo de datos actual, mediante una base de datos no relacional. Esta base de datos, facilitaría el trabajo con los datos debido a la naturaleza de los mismos. Rebajando la complejidad a la hora de poder extraer datos, teniendo que hacer relaciones entre las tablas de la base de datos.

- Mejorar el sistema de cuotas, creando diferentes planes para que los usuarios pudieran escoger que capacidad de datos desean, que numero de dispositivos o experimentos. Actualmente es infinito sin

ningún tipo de incentivos que pudieran provocar que la plataforma creciera en número de usuarios.

- Mejorar la seguridad tanto de la aplicación como de las comunicaciones, tratando de solventar lo posibles futuros agujeros de seguridad que pueden aparecer debido al buen hacer de atacantes.

# Bibliografía

[1] R. Dube, «Make use of,» Julio 2015. [En línea]. Available: http://www.makeuseof.com/tag/origins-wikipedia-si-title/.

[2] S. Leibson, «EDN Network,» 28 Marzo 2008. [En línea]. Available: http://www.edn.com/electronics-blogs/other/4306822/IPV6-How-Many-IP-Addresses-Can-Dance-on-the-Head-of-a-Pin-.

[3] Z. Y. X. Z. Bin Guo, «From Participatory Sensing to Mobile Crowd Sensing,» de *From Participatory Sensing to Mobile Crowd Sensing*, Xi'an, P. R. China.

[4] F. Y. a. H. L. Raghu K. Ganti, «Mobile Crowdsensing: Current State and Future Challenges,» *The Internet of things.*

[5] «Reactive Manifesto,» 16 Septiembre 2014. [En línea]. Available: http://www.reactivemanifesto.org/es.

[6] A. team, «Akka web page,» [En línea]. Available: http://doc.akka.io/docs/akka/2.4.4/intro/what-is-akka.html.

[7] O. Company, «Oracle Web page,» [En línea]. Available: http://docs.oracle.com/javaee/6/tutorial/doc/bnafe.html.

[8] R. T. Fielding, «Architectural Styles and the Design of Network-based Software Architectures,» 2000. [En línea]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

[9] M. Grönroos, Book of Vaadin, Lulu. com., 2011.

[10] D. Hardt, The OAuth 2.0 authorization framework, 2012.

[11] P. S. K. &. R. S. Mell, «A complete guide to the common vulnerability scoring system version 2.0.,» FIRST-Forum of Incident Response and Security Teams, 2007, pp. 1-23.

[12] Z. K. J. B. M. &. H. J. A. Durumeric, «Analysis of the HTTPS certificate ecosystem. In Proceedings of the 2013 conference on Internet measurement conference,» ACM, 2013, October, pp. 291-304.

# Anexo

So many years ago, some experts had a idea. They wanted to create a big sensor's network but was rejected for the high computational cost and the great space that these sensors occupy. Now, that idea can be re-lived because technology has reduced that costs and space with mobiles or Raspberries.

Those idea has been converted to actually Crowdsensing's platforms where users yield his sensors to create a community. At present, there are so many of this platforms, some has been stimulated by big companies. Every company uses his own communication's network with special messages. And it's impossible to change the period of extraction and the location of this.

Pervasive's group of the University Carlos III, developed a platform where it's possible to change some configuration of the extraction. It's possible to select which sensors will be in the experiments, too. The goal of this platform is that users can visualize data of this platform, it made necessary to develop a front-end. This front-end create a connection with the Clearing House to receive values and state of sensors.

This document contains the original design of the front-end (internal and visual) and the implementation of this design creating a prototype with a basic function.

This project is integrated inside a Crowdsensing's platform developed by the Pervasive's group of the Telematic department in the University Carlos III. The target of this project is the creation of a complete front-end in which one user can communicate with the platform to visualize the results of experiments that they have developed.

A few principal aims were established:

- ✓ Study of the currents Crowdsensing's platforms.

- ✓ To investigate communications to search a standardization in the existing platforms.
- ✓ To design a simple application which one users can check all data of different experiments which has been created or create a new of this.
- ✓ To depurate app and all the communication.
- ✓ To redact a document where explain the design and the implementation. This documents will have a chapter with a futures lines to continue the project.

This project was planned to do in 4 months. There were 5 task to complete, each one must be done before the previous was done.

| Num | Task | Days | First day | Final day | Previous task |
|---|---|---|---|---|---|
| 1 | Collect information | 25 | 01/02/2016 | 04/03/2016 | - |
| 2 | Design | 21 | 07/03/2016 | 04/04/2016 | 1 |
| 3 | Implementation | 30 | 05/04/2016 | 13/05/2016 | 2 |
| 4 | Test | 6 | 16/05/2016 | 23/05/2016 | 3 |
| 5 | Writing document | 16 | 24/05/2016 | 17/06/2016 | 4 |

**Tabla 2. Tareas planificadas (inglés)**

Task number 1 was one of the more importante because It was necessary to know how platforms work. When all information was collected, it's was possible to continue with the project. Any standardization was not found in this search. Each company uses his own way to communicate with the platform for this reason, it was necessary to create a new type of messages and a new communication.

Platform created by Pervasive's group has different levels which form the complete platform:

1. Physic level: this level is structured by different sensors that mobiles have.

2. Mobile level: is formed by mobiles and Raspberrys who collect information that sensors give to these devices.

3. Third level: It's the level where all information is modified to be sent.
4. Network level: It's the level what communicate the Clearing house and users.
5. User level: It's where information is showed to all users of platform.

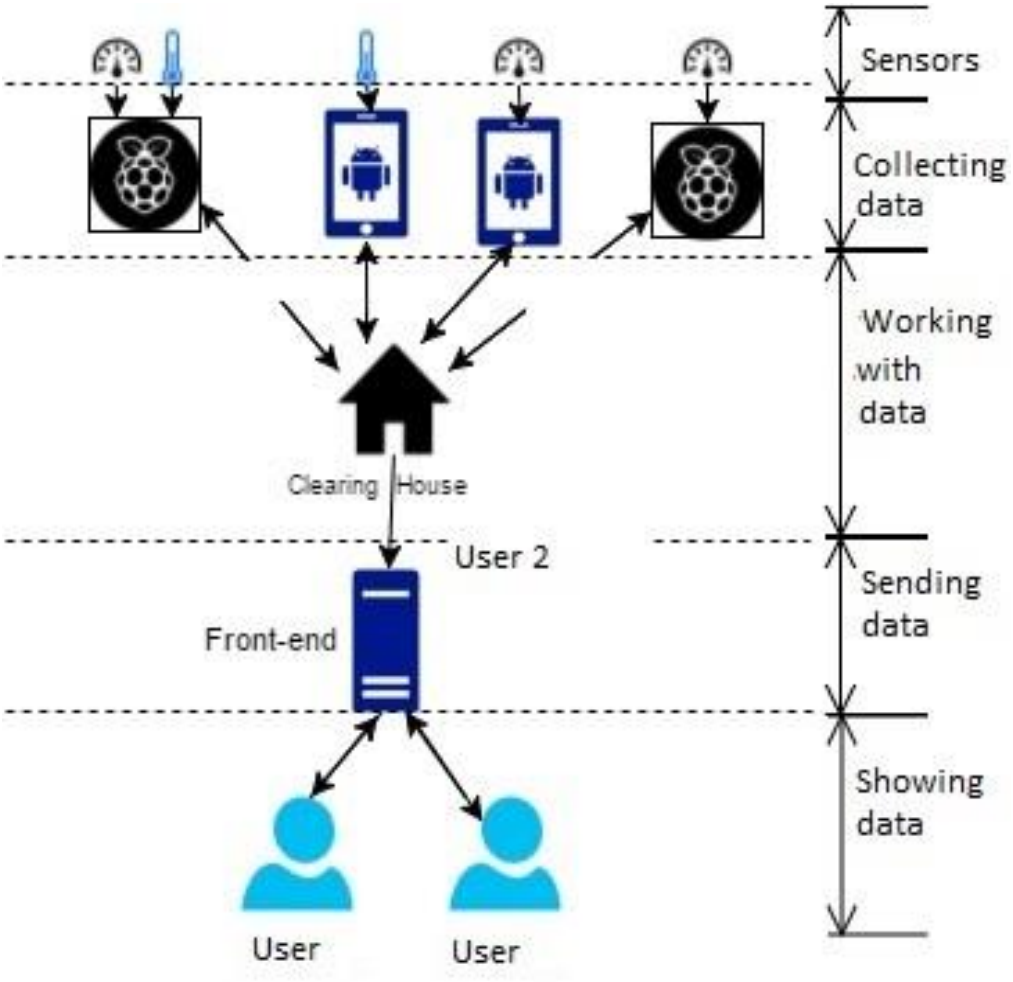


**Ilustración 14. Niveles plataforma Crowdsensing (inglés)**

This project is focus on fourth and fifth level, creating a complete front-end.

The Clearing House send to the front-end automatic requests to send information about the platform.

The original idea was that the front-end asks about information every time some user wants something but some test gave negative information about this idea. Finally, Clearing House sends each prefixed time. The front-end receives this information and saves it in the database.

The database has a few tables that contain all data: about user's personal information, experiment's information, sensor's information, etc. The information of the application is saved in a SQLite's database that persist the data model that was designed at the beginning. That kind of database has a high access speed and a high processing capacity. That features improve the use of the application.

This platform has a singular feature: users can create their own experiments. This feature makes necessary to enable a option in the application to upload the code of these experiments. The front-end sends this code to the Clearing House who floods the platform with the code. Users can see information about their experiments: a sensor is down, a device has a strange value or similar situations.

The front-end has two ways of receiving request from Clearing House. Spring's framework is waiting request to the modules: log and information. When a request has one of these modules like a target, the url of the request has "/log" or "/disp". When the front-end receive one of this url send the request to Spring, if the request doesn't have this path, the request will be send to Vaadin.

Spring has been configurated to read what is the path and ejecute a logic application. When a request is sent with this url, the request has some information to be saved in the database. This function is made by the Spring's framework.

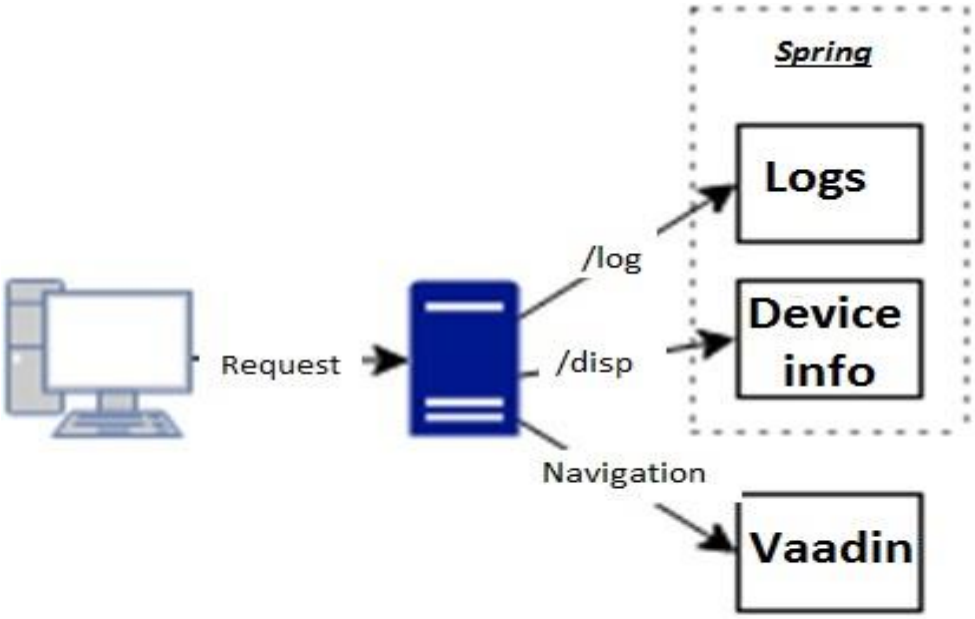


**Ilustración 15. Esquema funcionamiento aplicación (inglés)**

The use of a framework called Vaadin makes very easy to use the application. The front-end was developed with Java language that makes the logic of the application and design easier. The design has clear lines with simple images. All information that users can see are in 3 clicks, no more. This feature avoid users have to spend a lot of time to find what they want.

Vaadin is a framework that allows to create a front-end only using Java language. This feature makes simpler the implementation of the design. Users can select in the launchpad some options like: view their  experiments, their information, all devices in the network, etc.

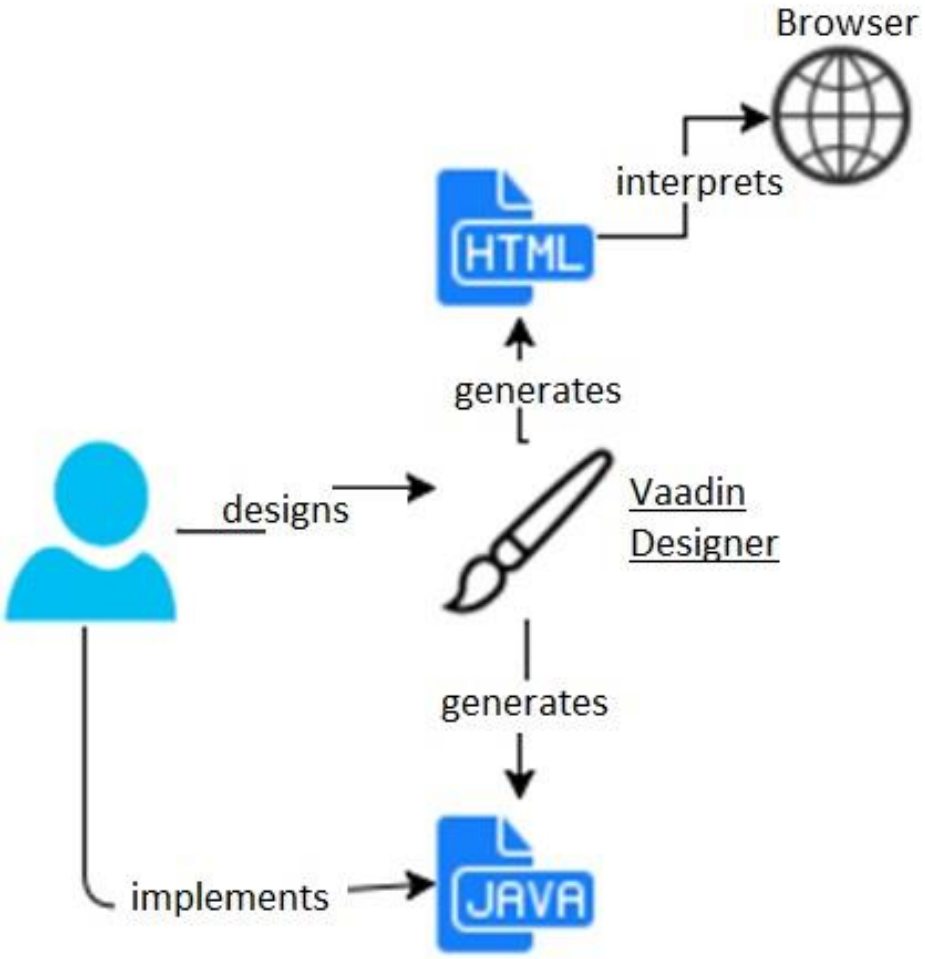


**Ilustración 16. Diagrama funcionamiento Vaadin Designer (inglés)**

The application allows to filter some list to make easier the use of the app. Users can filter devices by his characteristics. In Java 8, this action is very simple with the stream's class. This class allows to create a stream of strings.

When the users type in the filter field, this streams is modified deleting strings that not be the same that the filter.

Some security configuration has been implemented to make stronger the application. It's necessary to keep safe this data because it's very sensitive, it's personal information. An attacker can know where a person lives or where a person works.For this reason, it's very important to strengthen all data in the database. The communications are very exposed, too. It's recommended to use a secure HTTP connection, with HTTPS (SSL or TLS) **[12]**.

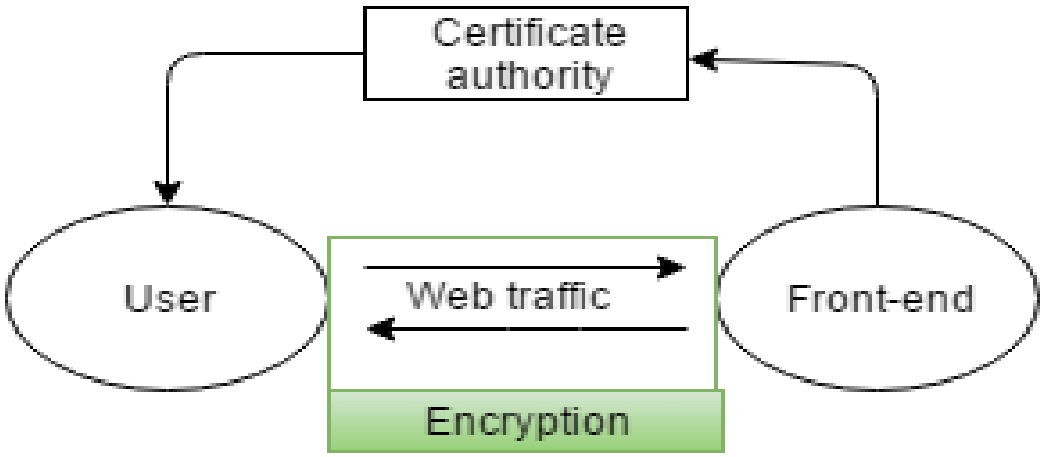


**Ilustración 17. Diagrama funcionamiento HTTPS**

As I mentioned some lines before, the project is a part of an existing Crowdsensing's platform has been commented. Nowadays, there are a few platforms with the same goal: users yield his mobiles with sensors to create a huge network where users can get different data of different locations. This platform has a special feature: users can create his own experiments to select where and when they want to extract this information. In others platforms, this

features doesn’t exist because it’s common to all users. To use this element it’s necessary the creation of a simple and intuitive application to connect platform with users. This users can visualize his own information and experiment’s information, too.

This platform has a clear objective: users should can see the information of sensors existing in the network. This view of data should be simple and easy to access.

This project has supposed a big effort caused by the standardization absence. This absence has a good notice: it was possible to implement the prototype with freedom. The platform has his own messages and the front-end implements a business logic to allow the creation of a better application.